NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DETECTING MALICIOUS TWEETS IN TWITTER USING RUNTIME MONITORING WITH HIDDEN INFORMATION

by

Abdullah Yilmaz

June 2016

Thesis Advisor: Doron Drusinsky
Second Reader: Man-Tak Shing

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY	**2. REPORT DATE** June 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE DETECTING MALICIOUS TWEETS IN TWITTER USING RUNTIME MONITORING WITH HIDDEN INFORMATION	**5. FUNDING NUMBERS** HDTRA139119
6. AUTHOR(S) Abdullah Yilmaz	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Threat Reduction Agency (DTRA) Ft. Belvoir, Virginia	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Although there is voluminous data flow in social media, it is still possible to create an effective system that can detect malicious activities within a shorter time and provide situational awareness.

This thesis developed patterns for a probabilistic approach to identify malicious behavior by monitoring big data. We collected twenty-two thousand tweets from publicly available Twitter data and used them in our testing and validation processes. We combined deterministic and nondeterministic approaches to monitor and verify the system. In the deterministic part, we determined assertions by using natural language (NL) and associated formal specifications. We then specified visible and hidden parameters, which are used for subsequent identification of hidden parameters in Hidden Markov Model (HMM) techniques. In the nondeterministic part, we used probabilistic formal specifications with visible and hidden parameters, used in HMM, to monitor and verify the system.

An important contribution of the work is that we specified some event patterns indicating malicious activities. Based on these patterns, we obtained output to indicate the possibility of each tweet being malicious.

14. SUBJECT TERMS formal specifications, hidden Markov model, hidden data, twitter, runtime verification, runtime monitoring, statechart assertions	**15. NUMBER OF PAGES** 77
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DETECTING MALICIOUS TWEETS IN TWITTER USING RUNTIME MONITORING WITH HIDDEN INFORMATION

Abdullah Yilmaz
Captain, Turkish Armed Forces, Army
B.S., Turkish Military Academy, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2016

Approved by: Doron Drusinsky
Thesis Advisor

Man-Tak Shing
Second Reader

Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Although there is voluminous data flow in social media, it is still possible to create an effective system that can detect malicious activities within a shorter time and provide situational awareness.

This thesis developed patterns for a probabilistic approach to identify malicious behavior by monitoring big data. We collected twenty-two thousand tweets from publicly available Twitter data and used them in our testing and validation processes. We combined deterministic and nondeterministic approaches to monitor and verify the system. In the deterministic part, we determined assertions by using natural language (NL) and associated formal specifications. We then specified visible and hidden parameters, which are used for subsequent identification of hidden parameters in Hidden Markov Model (HMM) techniques. In the nondeterministic part, we used probabilistic formal specifications with visible and hidden parameters, used in HMM, to monitor and verify the system.

An important contribution of the work is that we specified some event patterns indicating malicious activities. Based on these patterns, we obtained output to indicate the possibility of each tweet being malicious.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
ATG	Automatic Test Generation
EMC	Execution-Based Model Checking
FM	Formal Method
FS	Formal Specification
GPS	Global Positioning System
HMM	Hidden Markov Model
ISIS	Islamic State of Iraq and Syria
JSON	JavaScript Object Notation
MC	Model Checking
NL	Natural Language
R4B	Rules4business
REM	Runtime Execution Monitoring
RM	Runtime Monitoring
RV	Runtime Verification
TP	Theorem Proving
UML	Unified Modelling Language

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I would like to thank my family for their love and support throughout my time at the Naval Postgraduate School.

I would like to express my gratitude to Prof. Doron Drusinsky for his knowledge, wisdom, and enthusiasm. He always encouraged me. I am very grateful for his commitment as an advisor and his patience.

I am also thankful to Prof. Man-Tak Shing for his suggestions and support in helping to complete this thesis.

Finally, I would like to thank the staff members of Thesis Processing Office for their help.

This research was funded by a grant from the U.S. Defense Threat Reduction Agency (DTRA).

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

1. Social Media and Twitter

The Internet has become indispensable in our daily lives. It allows us to create and maintain communication and collaboration; through the use of social media and Twitter we can share everything from special milestones to routine daily activities.

According to statistics [1], there are nearly 3.010 billion active Internet users, which is nearly 41% of the world population, and 2.078 billion active social media accounts. The annual growth is 21% for Internet users and 12% for active social media accounts (Figures 1 and 2).

Figure 1. Snapshot of the World's Key Digital Statistical Indicators. Source: [1].

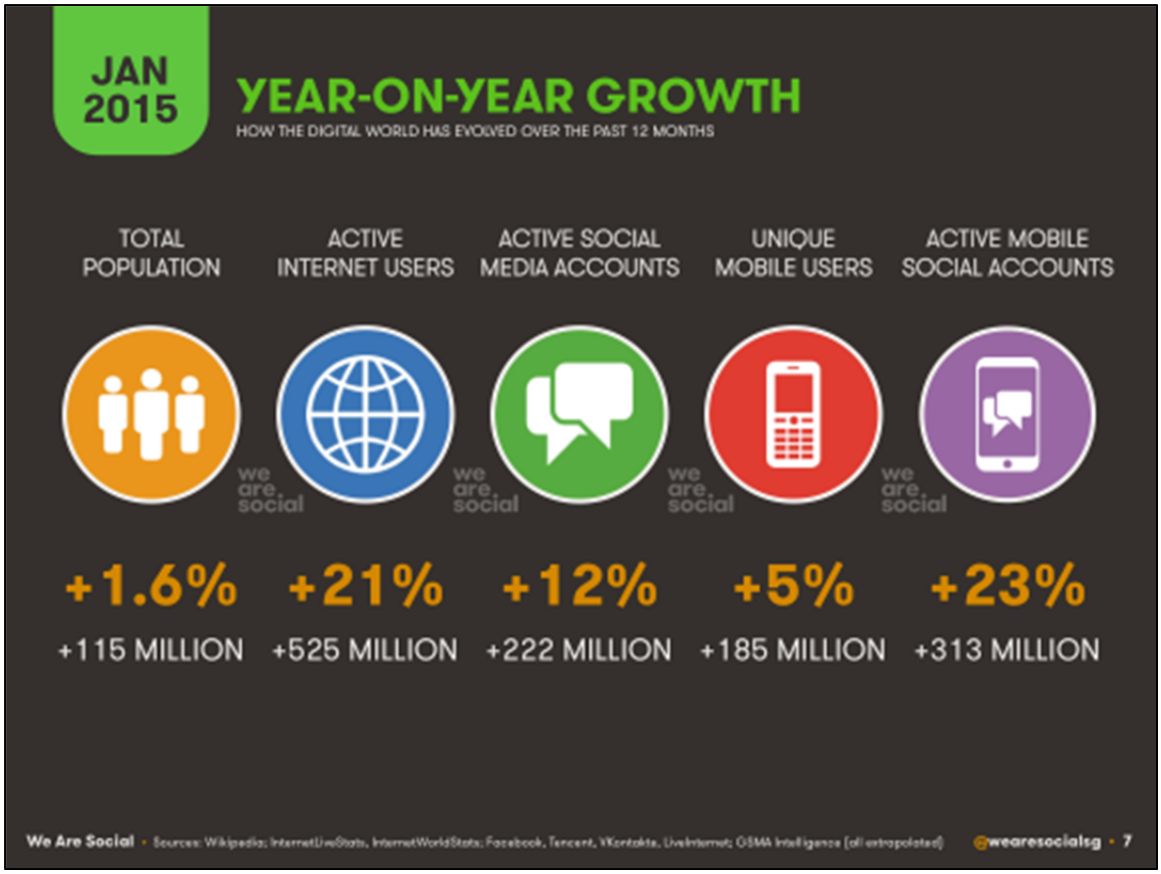


Figure 2. Annual Growth of the Digital World. Source: [1].

Twitter is a social networking service on the Internet. Twitter allows registered users to compose and send short messages of up to 140 characters, called tweets, to provide interconnection between users [2]. It also allows unregistered users to read tweets sent by others. We have focused on Twitter in this thesis since it is one of the biggest social media sites with 645,750,000 registered users [3] and has open source public tweets for data mining.

2. Malicious Users and Tweets

In the modern world, we face a new generation of terrorists, such as ISIS, who use social media, especially Twitter, to recruit new members. Propagandists can spread up to 200,000 tweets per day, using the platform as a propaganda tool with videos and pictures to feed the bad feelings of their supporters, spread fear to innocent people, and

trigger a series of lone-wolf terrorist attacks on orders from a terrorist leader on another continent [4].

Social media is a convenient place for malicious users for several reasons:

1. Detection is easily avoided.
2. Many people can be accessed with little effort.
3. Users in social media can share personal information such as credit cards, passwords, and private data.
4. It is easy to manipulate people by using popular contents, honeypots, and familiar accounts.

According to [5], Twitter has suspended nearly 125,000 accounts in the seven months leading up to February 2016 due to terrorist acts. Twitter has been reducing the response time for detecting malicious users and tweets and suspending the accounts with newly recruited staff. However, Twitter said that hunting terrorists on the web is not so simple since there is no “magic algorithm” to detect terrorist content.

In [6], Berger and Morgan give exhaustive information about the features of ISIS-supporter accounts, their tweeting patterns, and other twitter metrics. It is possible to use some of these metrics in our thesis for specifying the malicious users’ patterns. They are:

1. For about one out of five ISIS-supporter accounts, the primary language is English (73% Arabic, 18% English, and 6% French). It is common for the tweet to have an English hashtag with Arabic content.
2. The average number of followers of these accounts is about 1,004 (higher than regular users). While the number of the followers for a typical user is 208, for celebrities it is in the millions. ISIS supporters, therefore, have more followers than ordinary users. It is very unlikely to see an ISIS supporter with more than 20,000 followers.
3. The top locations of the accounts are Syria, Iraq, and Saudi Arabia. The location information of the accounts is very important for detecting these accounts. However, only a few users have enabled the location features. In addition, some of the users are using other applications to distort the GPS coordinates in creating the tweet.

4. Botnets and applications have been used to propagate a large number of tweets with specific content.

5. According to the statistics, nine out of ten ISIS-supporter accounts are following less than 1000 users. It is not a better indicator than follower information, but it can be useful to increase precision [6].

According to [6], we make some assumptions to help us identify malicious tweets:

1. The owners of the regular tweets have longer account lives than malicious users who tend to close their accounts for reasons such as hiding their real identity or accessing different users. Moreover, malicious accounts can be closed by Twitter after malicious behaviors have been detected.

2. The users spreading malicious tweets have more friends than followers since they want to access more people; however, they are not known by others in the network.

3. The accounts do not enable the geo location.

4. While a high number of followers (>20,000) indicates celebrities, a low number (<500) indicates regular users. Suspicious users are usually between them.

5. Malicious users follow less than 1000 users.

B. PURPOSE OF THE STUDY

Social media is a rapidly growing and changing space. It is a data pool, in that a wide variety of information can be captured as long as one knows how. This situation encourages us to adopt a new technique for the detection of security-related malicious activities. In this thesis, we will use a hybrid of probabilistic and formal methods to detect malicious activities.

C. THESIS STRUCTURE

Chapter I provides the motivation for this thesis. It explains the importance of social media in our lives. Chapter II provides background information on formal methods, formal specifications, the Hidden Markov Model, runtime monitoring, and verification. This chapter is important for understanding a new system based on these steps of the development cycle. Chapter III describes the steps for collecting, filtering,

and reforming the data acquired from Twitter. It provides useful information for those who want to data mine in Twitter, and presents the natural language assertions and corresponding rule patterns. It then describes the steps performed using screenshots from the toolset. The last chapter, Chapter IV, addresses thoughts and implications regarding the new technique.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. NATURAL LANGUAGE REQUIREMENTS

The typical software development process consists of requirement, design, and implementation stages. Before implementing any type of formal method, software developers try to present what they understand in terms of requirement and environmental arguments using natural language. Requirements are essential for a proposed system, since the system development cycle is cumulative in manner. If a developer misses a requirement, it can cause a heavy cost to fix the error. When stakeholder and system needs are specified as requirements in natural language, system developers should translate them into formal specifications for solving the problem in a systematic manner [7].

One may ask, “If we need to turn requirements into formal specifications, why are we using natural language in the initial stage?” (As done in this project.) The answer is that natural language is necessary because stakeholders, customers, or prospective users probably do not understand the formal specifications. Another reason is that nobody wants to sign a contract written in formal specification language [8]. So, natural language specification is vital for ensuring that everyone is on the same page.

B. FORMAL METHODS AND FORMAL SPECIFICATIONS

The growing use of software-intensive systems has increased the complexity of software development. This complexity multiplies the likelihood of errors and increases development cost. The major goal of software development is to develop reliable, efficient software-intensive systems despite the growing complexity. At this point, formal methods (FM) depending on a well-designed mathematical structure are very useful for solving the problem and providing precise implementations. FMs are a general collection of techniques including formal specification (FS) and program verification [9].

FS is a technique based on a mathematical structure. It consists of syntax for grammatical rules and semantics for interpretation [10]. The purpose of the FS is to

clearly represent a cognitive or natural language requirement and make it easy to monitor system behaviors [11].

The use of formal methods and specifications has changed over time as a result of the growing complexity of systems. Because full formalization of systems has become more expensive and difficult, some scientists have proposed lightweight formal methods focused on partial specification and application [12].

Lightweight formal methods are very useful for reducing development cost. Moreover, FSs and lightweight FMs improve the clarity and precision of specified requirements [13]. Runtime execution monitoring (REM) is a lightweight FM that monitors the behavior of a running system and can detect improper behavior and requirements in early stage of development. Debugging of the requirements and early detection of the errors in the design process prevents extra cost and time.

In [13], Drusinsky presents a new type of formal method using a combination of runtime monitoring, execution-based model checking, and UML-based formal specifications with statechart assertions that provide unambiguous, clear, and visual presentation of the model. This new formalism uses deterministic/nondeterministic statechart assertions as its specification language [13]. Figure 3 shows a statechart assertion in which there is a start state, final state, event state, timer, and transitions between them.

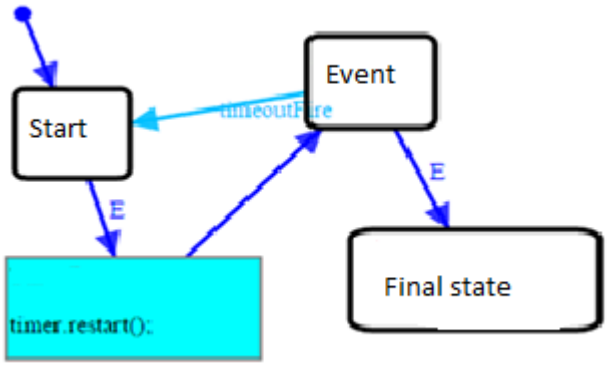


Figure 3. A Statechart Assertion.

C. DETERMINISTIC RUNTIME VALIDATION AND VERIFICATION

The correctness of a system is directly related to the validation and verification of the system. Verification checks whether the system produces correct results given its input. It does so by comparing the expected output to the actual output. If there is an inconsistency, it means that verification failed. Validation looks for whether the system meets our intended purposes. Is it the right product to build? While the verification process focuses on the internal parts such as behaviors of the system, the validation process checks the overall system as a final product and compares to the intended product [7].

D. FORMAL VERIFICATION AND VALIDATION TRADEOFF SPACE

As Drusinsky et. al. pointed out in [7], there are cost and coverage space tradeoffs for verification and validation (V&V) techniques. We will describe the tradeoffs by using two 3-dimensional (3D) cuboids. These vectors are specification/validation, program/implementation, and verification. Figures 4 and 5, respectively, represent the coverage space and cost space tradeoffs of V&V techniques.

Three types of V&V techniques are: theorem proving (TP); model checking or property checking (MC); and execution-based model checking (EMC) combining runtime verification (RV) with automatic test generation (ATG).

1. Theorem Proving

High Order Logic (HOL) and Stanford Temporal Prover (STeP) are the methods using TP. TP employs mathematics-based proof techniques to provide a persuasive argument that demonstrate that a program complies with its requirements. This technique requires a human driver to solve the underlying problem which is generally undecidable.

In respect to cost and coverage tradeoffs, an important aspect of TP is that a human operator whose skill level changes according to the choice of the specification language is required [14]. In TP, specification languages such as temporal logic are generally difficult to understand and use. These languages are hard to implement since they are different from the languages that software programmers use. It is difficult to

validate formal specifications with limited knowledge about temporal logic syntax. So, it has low specification coverage and high specification cost. TP depends on the programming languages that contain many inconsistencies with the existing Java or C++ applications. So it deserves the low program coverage and high implementation cost in terms of the program/implementation dimension. While the expertness requirement of TP causes high verification cost, the existence of well-educated and wise users provides high verification coverage [7].

2. Model Checking

MC is a kind of algorithmic FV technique that checks the state space of the model exhaustively for whether it satisfies the given specifications. Some of the published MC tools are SPIN Model Checker and Spatial Logical Model Checker (SPML) verifying the correctness of the distributed model.

When a program is set up for MC, there is no need for an expert human operator. So, contrary to TP, human expertise is not required in MC. It has a lower verification cost than TP. Both TP and MC have text-based specifications, which causes difficulty in visualization and validation processes for the designer. Consequently, MC deserves low specification coverage similar to TP [7]. With respect to the program/implementation dimension, the most vulnerable point of this verification technique is the blowing up of the state-space (a.k.a. combinatorial explosion) problem. This problem typically causes the verification process to be crushed by an exponentially growing state-space [15]. Eventually, while the technique deserves low coverage and high cost for program/implementation dimension, it has high coverage and low cost for the verification dimension because of the automatic verification and full coverage of components.

3. Execution-Based Model Checking (EMC)

Runtime verification (RV) and automated test generation (ATG) are components of EMC. Some of the RV tools are DBRover and StateRover that include statechart diagrams as specification language. StateRover also has an automatic test generator.

In runtime verification, the system is monitored while running and generating tests by ATG. With this technique, UML-based StateRover specification language—a dynamic, lightweight V&V tool—can be used. It has automation and can cope with large, complicated systems. Hence, it has high coverage and low cost for the specification and program/application dimensions. Reliability to ATG is the one weakness of this technique; it is possible to miss a violation that ATG cannot generate a suitable test. So, ATG provides low cost and intermediate coverage for the verification dimension [15]. Although RV has less coverage than conventional verification techniques like MC and TP, it keeps us away from the complexity of the verification process [16].

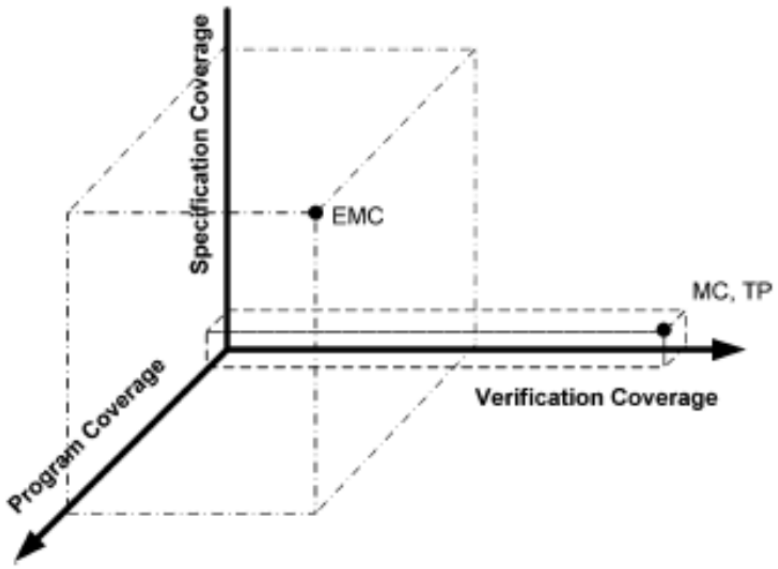


Figure 4. Coverage Space. Source: [7].

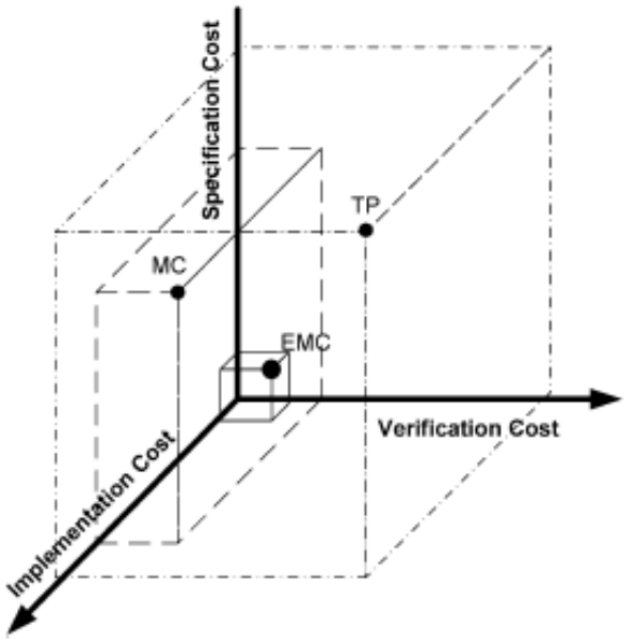


Figure 5. Cost Space. Source: [7].

According to the comparison of three V&V techniques, EMC has lower implementation, verification, and specification costs. It has higher program and specification coverages.

Note that we do not verify any software systems in our thesis. We use a powerful FS language with runtime monitoring in EMC since runtime monitoring satisfies our needs for situational awareness, which is a prerequisite for stable and reliable systems. The method ensures more consistent and trustworthy results for categorization of tweets in a running system.

E. HIDDEN MARKOV MODEL (HMM)

Markov Models are stochastic models that are used in randomly altering systems. They have a list of possible states. Each present state has possible future state(s). There are four main Markov Models used in various problem areas [17]. They are the Markov chain, the Markov decision process, HMM, and the partially observable Markov decision process (Table 1).

Table 1. Markov Models. Source: [17].

	System state is fully observable	**System state is partially observable**
System is autonomous	Markov chain	Hidden Markov model
System is controlled	Markov decision process	Partially observable Markov decision process

In simpler Markov models such as the Markov chain, there is only one parameter—"state transition probabilities"—and states are fully observable. In HMM, the states are not fully visible and each state has possible observations, state transition probabilities, and output probabilities [18].

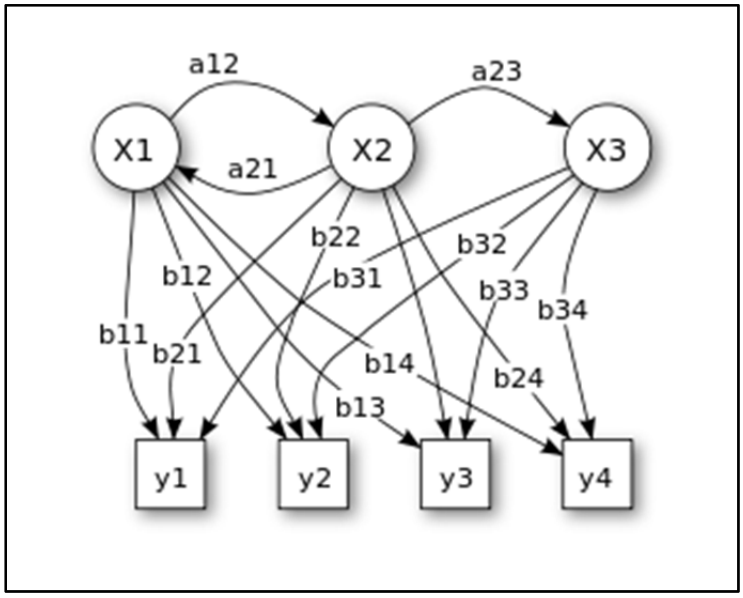


Figure 6. Markov Model. Source: [18].

In Figure 6, X, y, a, and b indicate states, possible observations, state transition probabilities, and observation probabilities, respectively.

An HMM is very similar to a state machine: both have states and transitions between states. Each transition between states and the observable of the states are assigned a probability value between 0 and 1. The model determines one of the possible

outputs by looking at the sequence of observables [19]. In our thesis, the hidden states of the model is the categorization of the tweets. We will look for patterns, which are seen as rules in Section 3, to interesting flag sequences of tweets.

F. RUNTIME MONITORING AND VERIFICATION OF SYSTEMS WITH HIDDEN DATA

Runtime monitoring (RM) is a technique for observing runtime system behavior. While doing so, it detects formal specification violations.

When applying RM and RV to complex systems, the required information such as the existence of malicious email or tweets are not fully observable. In [20], Drusinsky presents a RM technique that can be implemented in a system, which includes hidden events. The technique uses UML-based statechart assertions; it combines HMM and RM of formal specification assertions [20].

In this study, we combine HMM with RM of statechart assertions, where the HMM is used categorize tweets as one of: malicious, suspicious or benign.

The flow chart of a system using RM with hidden data processing is shown in Figure 7.

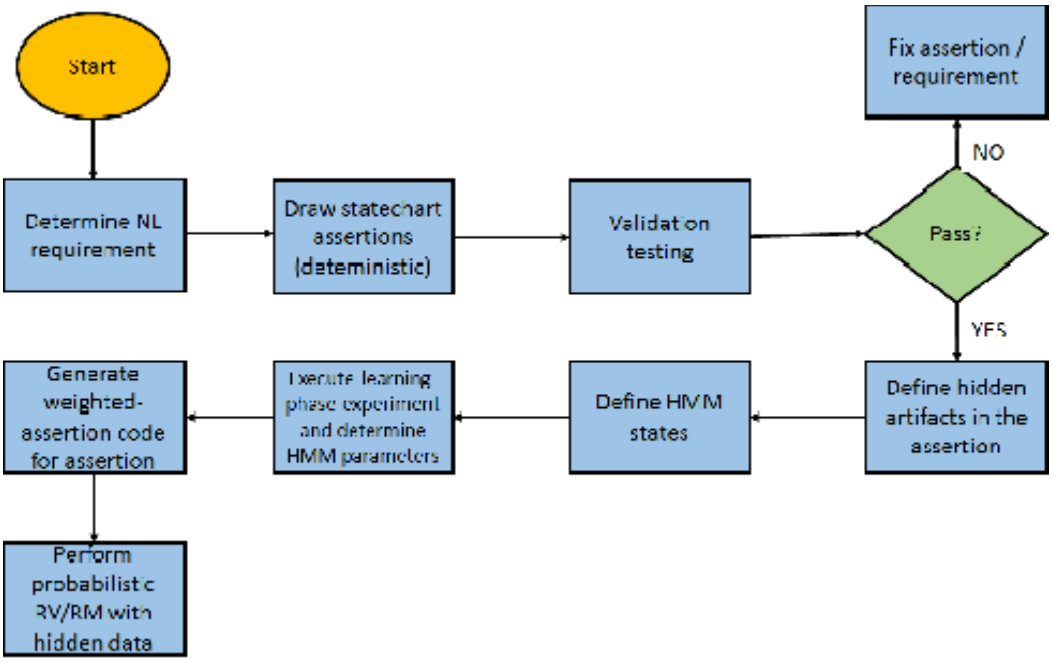


Figure 7. Flow Chart. Adapted from [20].

THIS PAGE INTENTIONALLY LEFT BLANK

III. RUNTIME MONITORING OF TWEETS

A. COLLECTING AND FILTERING DATA

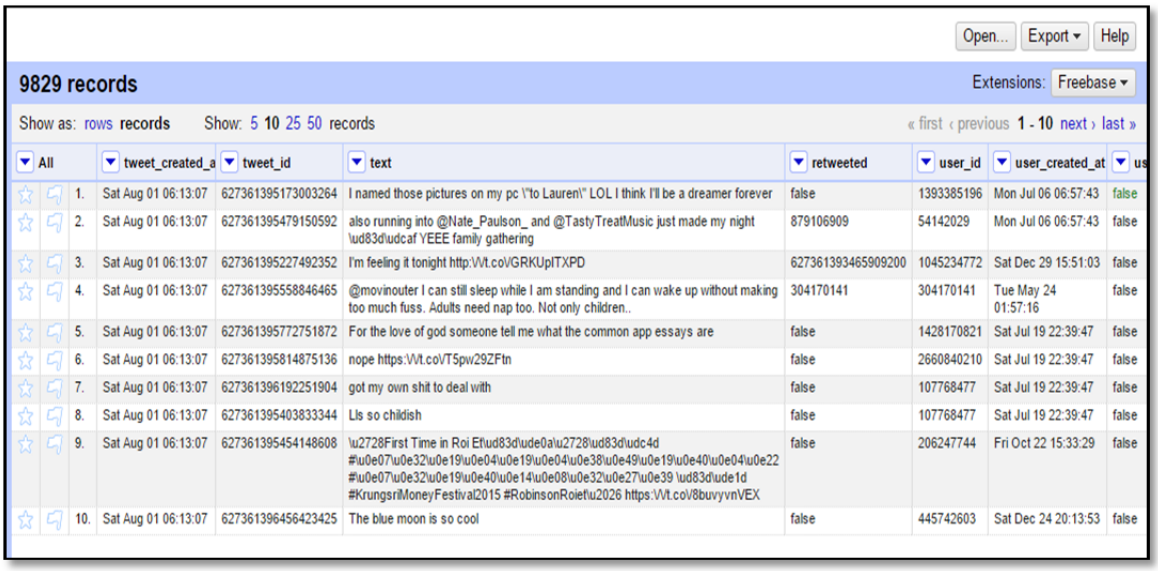
The Twitter Streaming API was used to collect data from Twitter. First, we created an account and generated tokens to be used as user credentials. Figure 8 shows the Python code used for downloading tweets [21]. We attached the unique user credentials provided by Twitter for the variables access_token, access_token_secret, consumer_key, and consumer_secret.

```
# Code modified from "Connecting to Twitter Streaming API and downloading data" code
# obtained from http://adilmoujahid.com/posts/2014/07/twitter-analytics/
import json
import time
import pandas as pd
import matplotlib.pyplot as plt

#Import the necessary methods from tweepy library
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener

#Variables that contains the user credentials to access Twitter API
access_token = "[redacted]"
access_token_secret = "[redacted]"
consumer_key = "[redacted]"
consumer_secret = "[redacted]"

class listener(StreamListener):
    def on_data(self,data):
        print(data)
        return True

    def on_error(self,status):
        print(status)
        return TRUE #do not kill the stream

    def on_timeout(self):
        return TRUE #do not kill the stream

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
twitterStream = Stream(auth, listener())
twitterStream.filter(locations=[-180,-90,180,90], languages = ['en'])
```

Figure 8. Python Code to Download Data from Twitter. Source: [21].

We collected 22,000 tweets from publicly available Twitter data in the JSON data structure. In this form, the data is not reader-friendly and is unsuitable for validation and formal specifications. Moreover, there are too many features for each tweet. Hence, it is

necessary to filter them for specific features of interest, such as when the user account was created, the number of followers and friends, the number of retweets, text, and so on. The data was converted into csv format and filtered by using the Google Refine tool, which is a powerful tool for messy data. Google Refine can filter the data and transform it to another format. Figure 9 shows a snippet of data including columns and records in Google Refine.



All		tweet_created_a	tweet_id	text	retweeted	user_id	user_created_at	us
1.	Sat Aug 01 06:13:07	627361395173003264	I named those pictures on my pc \"to Lauren\" LOL I think I'll be a dreamer forever	false	1393385196	Mon Jul 06 06:57:43	false
2.	Sat Aug 01 06:13:07	627361395479150592	also running into @Nate_Paulson_ and @TastyTreatMusic just made my night \ud83d\udcaf YEEE family gathering	879106909	54142029	Mon Jul 06 06:57:43	false
3.	Sat Aug 01 06:13:07	627361395227492352	I'm feeling it tonight http:\/\/t.co\/GRKUpITXPD	627361393465909200	1045234772	Sat Dec 29 15:51:03	false
4.	Sat Aug 01 06:13:07	627361395558846465	@movinouter I can still sleep while I am standing and I can wake up without making too much fuss. Adults need nap too. Not only children..	304170141	304170141	Tue May 24 01:57:16	false
5.	Sat Aug 01 06:13:07	627361395772751872	For the love of god someone tell me what the common app essays are	false	1428170821	Sat Jul 19 22:39:47	false
6.	Sat Aug 01 06:13:07	627361395814875136	nope https:\/\/t.co\/T5pw29ZFtn	false	2660840210	Sat Jul 19 22:39:47	false
7.	Sat Aug 01 06:13:07	627361396192251904	got my own shit to deal with	false	107768477	Sat Jul 19 22:39:47	false
8.	Sat Aug 01 06:13:07	627361395403833344	Lls so childish	false	107768477	Sat Jul 19 22:39:47	false
9.	Sat Aug 01 06:13:07	627361395454148608	\u2728First Time in Roi Et\ud83d\ude0a\u2728\ud83d\udc4d #\u0e07\u0e32\u0e19\u0e04\u0e19\u0e04\u0e38\u0e49\u0e19\u0e40\u0e04\u0e22 #\u0e07\u0e32\u0e19\u0e40\u0e14\u0e08\u0e32\u0e27\u0e39 \ud83d\ude1d #KrungsriMoneyFestival2015 #RobinsonRoiet\u2026 https:\/\/t.co\/8buvyvnVEX	false	206247744	Fri Oct 22 15:33:29	false
10.	Sat Aug 01 06:13:07	627361396456423425	The blue moon is so cool	false	445742603	Sat Dec 24 20:13:53	false

Figure 9. A Snippet of Data in Google Refine.

B. MEANING OF THE DATA COLUMNS

We used the variables shown as columns in Figure 10. The meaning of each column is defined in Table 2.

user_created_at	text	followers_count	friends_count	user_verified	geo_enabled
2009-03-13	Super Lol https:\/\/t.co\/1rj3JaAEjn	2946	1506	FALSE	TRUE
2009-04-18	@BluthX @joanwalsh How is stating fa	219	740	FALSE	TRUE
2009-06-03	23. I use to be the captain of SCTCC LYM	204	149	FALSE	TRUE
2010-01-14	The seats are being filled ahead of the	265880	420	TRUE	TRUE
2010-05-15	@staceymurdough1 thanks chick! I'll ta	1226	828	FALSE	TRUE
2010-05-18	@The_Gatorr get on damn lol	695	2	FALSE	FALSE
2010-07-12	@TT_Sisters @LittleMeThatter @GaryB	204	149	FALSE	FALSE
2010-07-13	@Theominiking \nYou was absolutely a	580	1	FALSE	FALSE
2010-07-19	talaye!http	935	24	FALSE	FALSE

Figure 10. A Snippet of Data Columns Used in Thesis.

Table 2. Meaning of the Columns.

Columns	Meaning
user_created_at	The date when the user account of the tweet is created
text	The messages sent through the Internet; the main body of the tweet.
followers_count	The current number of followers for this account.
friends_count	The number of users that this account is following.
user_verified	If set to True, then Twitter has officially certified the user's identity.
geo_enabled	If set to True, the user has enabled Twitter to access location information.

In Table 3, we classified all variables as alpha and beta columns since we use different columns for some steps in our work flow. For example the user_verified and geo_enabled columns are only used in the learning phase of the work flow.

Table 3. Alpha and Beta Columns.

Type	Column Name	Stage to being implemented
Alpha columns	followers_count	These columns are used in the learning phase for determining the HiddenState column.
	friends_count	
	user_verified	
	geo_enabled	
Beta columns	user_created_at	These columns are used in the R4B and RM phases.
	text	
	followers_count	
	friends_count	

C. NATURAL LANGUAGE ASSERTIONS AND DETERMINISTIC RULE DEVELOPMENT

The first step for development of a RM system for tweets is to specify requirements by using natural language (NL) and determine corresponding formal specifications (FS) [10]. We use Rules4business (R4B) for formal specifications. The requirements in NL are expressed by patterns provided in R4B.

For determining malicious tweets, we made some assumptions based on Chapter I (Motivation), as follows.

The owners of the regular tweets have a longer account life than malicious users who tend to close their account for reasons such as hiding their real identity or accessing different users. Moreover, malicious accounts can be deleted by Twitter.

In [22], Shingh, Bansal, and Sofat point out that malicious users reach a large number of friends in a short time and use popular links to spread their tweets faster and attract attention. On the other hand, famous users in Twitter have a larger number of followers and smaller number of friends. Therefore, we can assume that a large number of followers (>20,000) indicates celebrity status, whereas low numbers (<500) indicate regular users. Suspicious users are usually between the two values in number of followers. On the other hand, malicious users follow fewer than 1000 users (friends<1000).

According to our inferences from [4], [5], and [6], malicious users' accounts are not verified by Twitter and the users often disable the geolocation option. Their account life is two years or less and they usually open new accounts *before* their current accounts are identified as malicious.

We use R4B to choose and customize our statechart assertions. R4B has two different interfaces for customization of instances and validation, respectively. In the first page, users select rules according to the NL assertions. They can create and edit instances. In the second page, in order to validate assertions, users upload the spreadsheet including the columns that each rule requires. These columns are our beta columns. We create five rules; they are instances of R4B rules: Rule-1, Rule-3, Rule-9, Rule-19, and Rule 21 as shown in Table 4. In Table 4, a pattern is called as a generic rule in R4B. "P=HiddenState=="M" and friends_count<1000 and followers_count<20000 and followers_count>500" means the event in which the tweet has more than 1000 friends. This tweet also is marked as malicious and its number of followers is between 500 and 20000. "HiddenState=="B"" and "HiddenState=="S"" indicate the tweets marked as

benign and suspicious, respectively. Figures 11 to 15 show the statechart assertions for generic rules called as patterns in Table 4.

Table 4. Instances of R4B Patterns.

Rules4business		
Rule-1	Pattern	Flag whenever event P happens.
	Events and Limits&Bounds	P=HiddenState=="M" and friends_count<1000 and followers_count<20000 and followers_count>500
	Description	Flag the tweet whenever there is a malicious (HiddenState) tweet with <1000 friends and >500 followers and <20000 followers.
Rule-3	Pattern	Flag whenever no event Q occurs between P and R.
	Events and Limits&Bounds	Q=HiddenState=="B", P=HiddenState=="M", R=HiddenState=="M".
	Description	Flag the tweet whenever there is no benign tweet between two malicious tweets.
Rule-9	Pattern	Flag whenever some pair of consecutive E events is less than time T apart.
	Events and Limits	E=HiddenState==="S", Time bounds: T=4, Time units: weeks
	Description	Flag the tweet whenever two users have suspicious (HiddenState) tweets and are created <4 weeks apart.
Rule-19	Pattern	Flag whenever more than N events E within time T after Q.
	Events and Limits&Bounds	E=friends_count<1000 and followers_count<20000 and followers_count>500, Q=HiddenState==="M" N=3, T=10
	Description	Flag the tweet whenever there are >3 tweets which are <1000 friends and >500 followers and <20000 followers, within 10 days after the user of a malicious (HiddenState) tweet created.
Rule-21	Pattern	Flag whenever event Q occurs >N times between some pair of consecutive E events
	Events and Limits&Bounds	E=text.indexOf("http")>=0, E=Q=HiddenState==="S" N=2 (count bounds)
	Description	Flag when there are >2 suspicious tweets between any two tweets including http link.

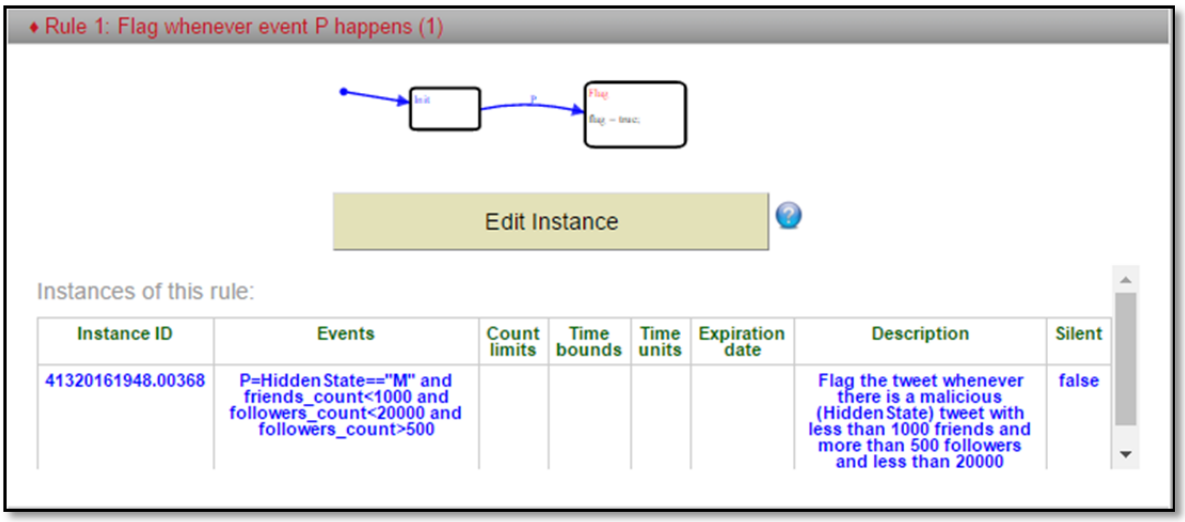


Figure 11. Instance of Rule-1 from R4B.

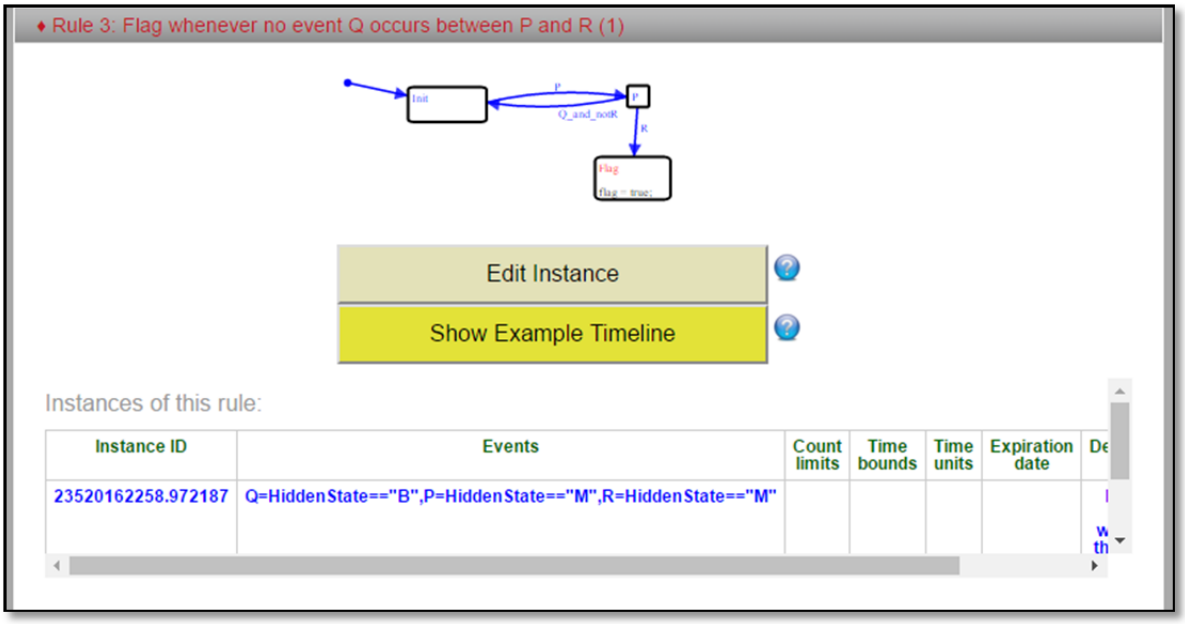
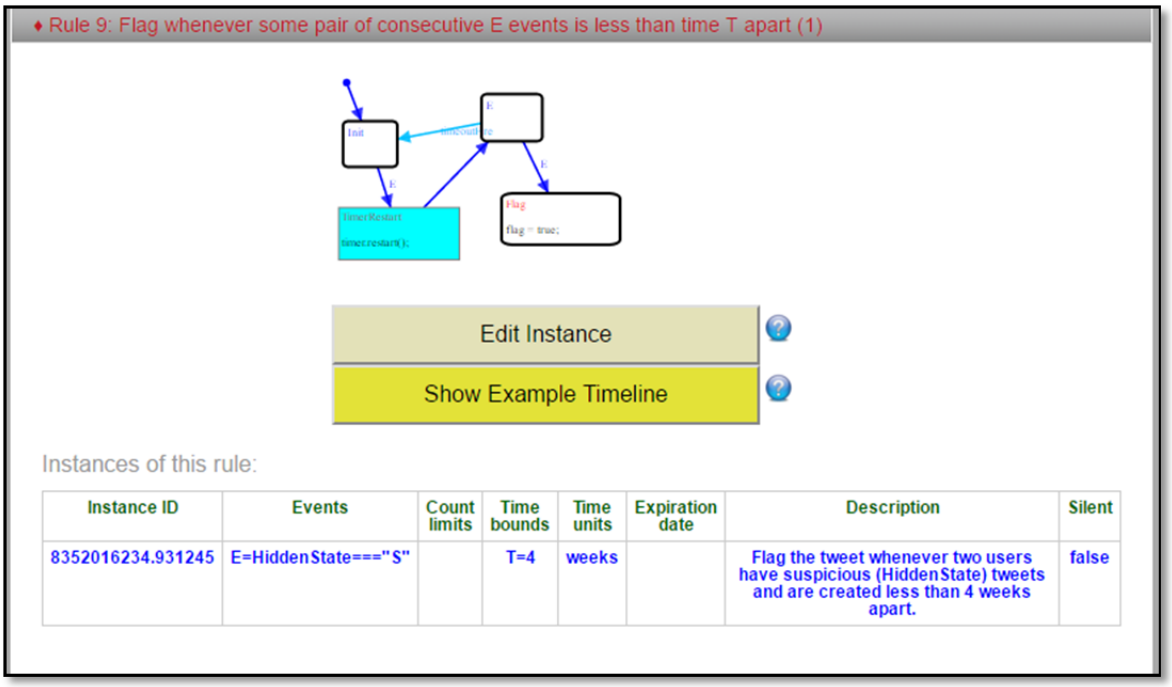
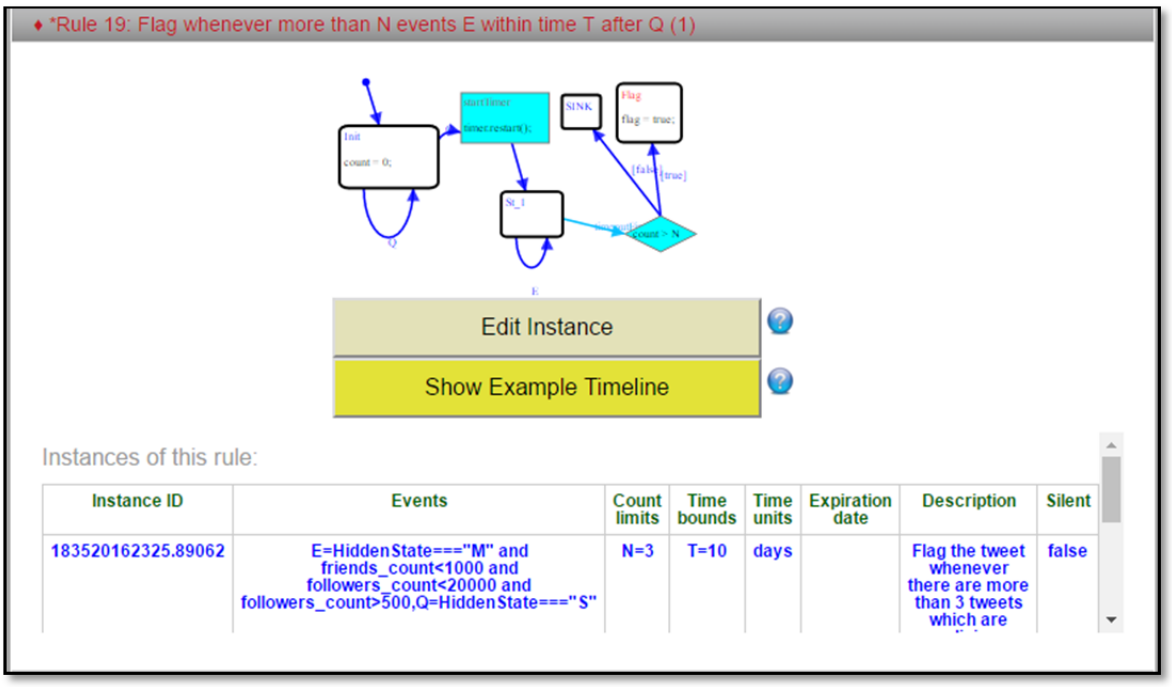


Figure 12. Instance of Rule-3 from R4B.



Instance ID	Events	Count limits	Time bounds	Time units	Expiration date	Description	Silent
8352016234.931245	E=HiddenState==="S"		T=4	weeks		Flag the tweet whenever two users have suspicious (HiddenState) tweets and are created less than 4 weeks apart.	false

Figure 13. Instance of Rule-9 from R4B.



Instance ID	Events	Count limits	Time bounds	Time units	Expiration date	Description	Silent
183520162325.89062	E=HiddenState==="M" and friends_count<1000 and followers_count<20000 and followers_count>500,Q=HiddenState==="S"	N=3	T=10	days		Flag the tweet whenever there are more than 3 tweets which are	false

Figure 14. Instance of Rule-19 from R4B.

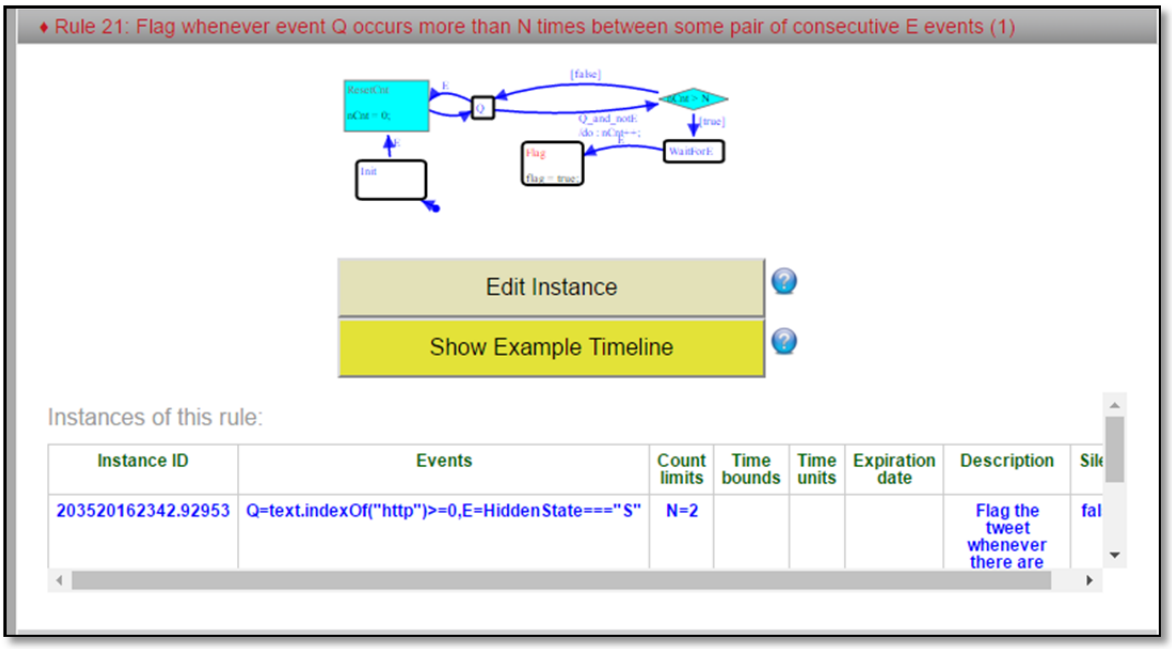


Figure 15. Instance of Rule-21 from R4B.

D. VALIDATION OF ASSERTIONS IN RULES4BUSINESS

We validated our assertions by uploading a file called “validation spreadsheet”. Before uploading such a csv or xlsx format spreadsheet (Figure 16), we specified column indexes as follows: “user_created_at=1, text=2, followers_count=3, friends_count=4, HiddenState=5, time=1.” Because in our spreadsheet the date of account creation, text part of the tweet, number of followers, number of friends, and state of tweet information are given in same order. For example, number of follower information is shown in third column. A time column represents the baseline (x-axis) of the flag timeline diagram.

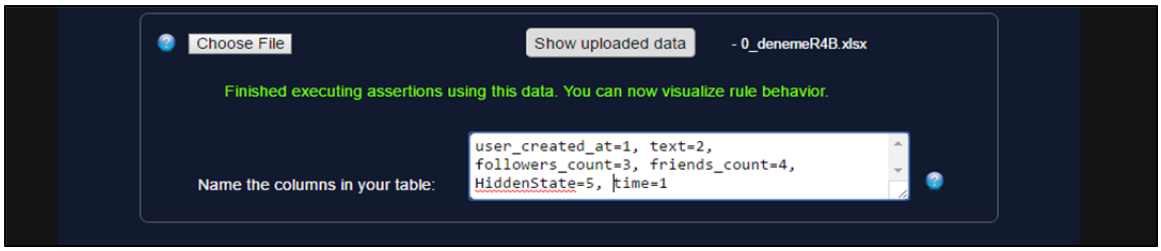


Figure 16. Naming the Columns in R4B.

In the validation phase, we use two different versions of validation spreadsheets as shown in Figures 17 and 18. The validation spreadsheet consists of all columns used in the R4B site and the HiddenState column. These spreadsheets have different values to induce flags. The highlighted cells in Figure 18 shows the differences between the two spreadsheets. Thus we can check the expected results as explained below. Figures 17 and 18 depict the first 12 rows of the validation spreadsheet.

user_created_at	text	followers_count	friends_count	HiddenState
2009-03-13	Super Lol https:\/\/t.co\/1rj3JaAEjn	2946	1506	S
2009-04-18	@BluthX @joanwalsh How is stating fa	219	740	S
2009-06-03	23. I use to be the captain of SCTCC LYM	204	149	S
2010-01-14	The seats are being filled ahead of the	265880	420	B
2010-05-15	@staceymurdough1 thanks chick! I'll ta	1226	828	S
2010-05-18	@The_Gatorr get on damn lol	695	2	M
2010-07-12	@TT_Sisters @LittleMeThatter @GaryB	204	149	S
2010-07-13	@Theominiking \nYou was absolutely a	580	1	M
2010-07-19	talaye!http	935	24	M
2010-07-19	Most amazing moment of 2016. Discuss	935	736	M
2010-07-22	@TRobinsonNewEra @lynbrownmp @	543	312	S
2010-07-22	You can stop it. Yes . Can stop it .	326	2	S

Figure 17. R4B Validation Spreadsheet Version 1

user_created_at	text	followers_count	friends_count	HiddenState
2009-03-13	Super Lol https:\/\/t.co\/1rj3JaAEjn	2946	1506	M
2009-04-18	@BluthX @joanwalsh http How is stating	219	740	S
2009-06-03	23. I use to be the captain of SCTCC LYM C	204	149	S
2010-01-14	The seats are being filled ahead of the sta	265880	420	S
2010-05-15	staceymurdough1 thanks chick! I'll take p	1226	828	S
2010-05-18	@The_Gatorr get http on damn lol	695	2	S
2010-07-12	@TT_Sisters @LittleMeThatter @GaryBarl	204	149	M
2010-07-13	@Theominiking \nYou was absolutely am	580	1	M
2010-07-19	talaye!http	935	24	M
2010-07-19	Most amazing moment of 2016. Discussing	935	736	M
2010-07-22	@TRobinsonNewEra @lynbrownmp @Grc	543	312	S
2010-07-22	You can stop it. Yes . Can stop it .	326	2	S

Figure 18. R4B Validation Spreadsheet Version 2

According to the first validation spreadsheet, we expect Rule-1 to induce an RM flag in rows 6, 8, 9, and 10. For Rule-3, rows 8 and 10 are expected to induce a RM flag. While for Rule-9 there is a single RM flag expected in row 11, we do not expect any flags for Rule-19 and 21.

We perturbed some values in the first validation spreadsheet (Figure 17) to create a second validation spreadsheet (Figure 18). For example, in row 8, we changed the HiddenState value from malicious (M) to suspicious (S). The user now has less than 1000 friends and his/her number of followers is between 500 and 20,000. It was induced an RM flag for Rule-1, but after changing the HiddenState value to suspicious, it is expected to not induce an RM flag; indeed, it did not—as depicted in Figure 19. Likewise, Rule 9 is expected to not induce an RM flag in row 6 because it was not a suspicious tweet; indeed, it did not—as depicted in Figure 20. However, when we changed the HiddenState column to suspicious, Rule 9 is expected to induce an RM flag because these two tweets were created three days apart and both of them were classified as suspicious; indeed, RM induced such a flag, as depicted in Figure 20.

This is the essence of the validation phase: to check that all rules induce an RM flag precisely when expected to do so.

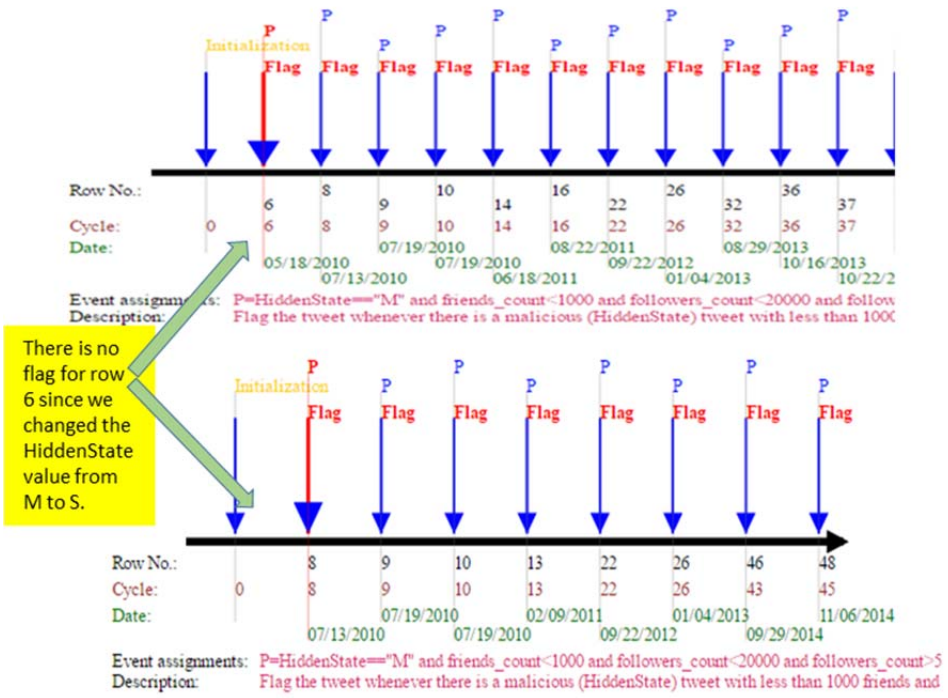


Figure 19. Rule-1 Flag Timeline Diagram

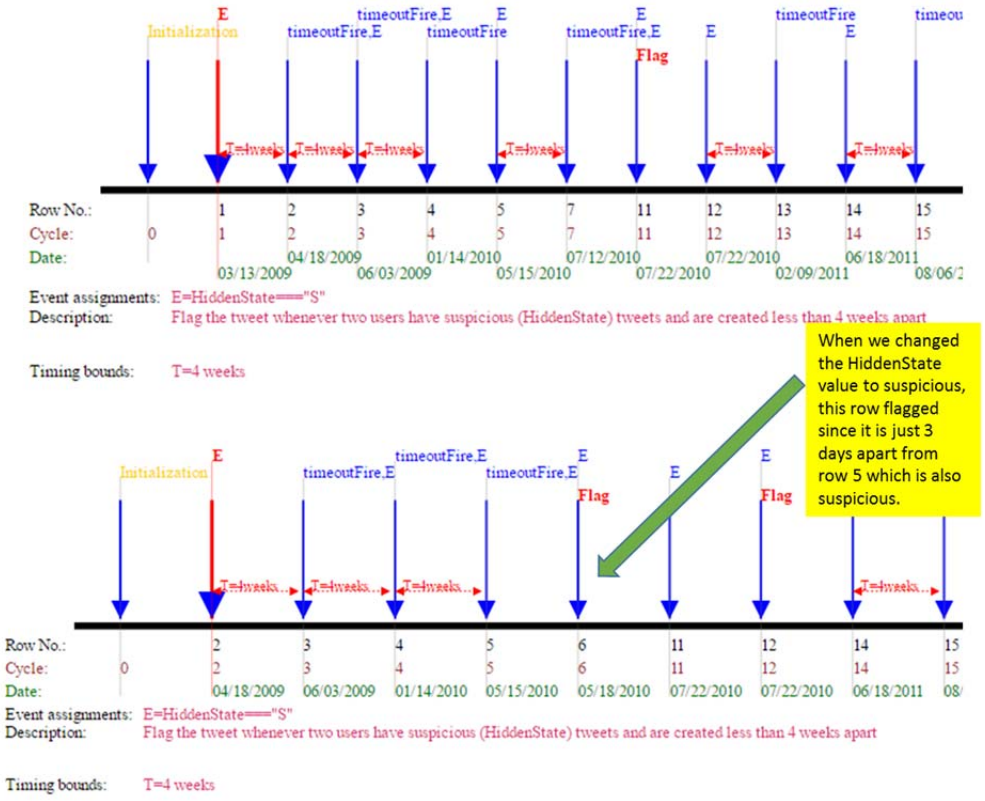


Figure 20. Rule-9 Flag Timeline Diagram

R4B presents visualization for behavior of each rule. Figure 21 presents this visualization for Rule 9. In Figure 21, the upper left window shows statechart diagram. Lower left window presents uploaded file with flagged tweet in row 11. Flagged tweets are displayed in red. The right window is the timeline diagram showing all flag and non-flag states in time axis. The tweet in row 11 is flagged since tweets in row 7 and 11 are both suspicious and they are less than four weeks apart.

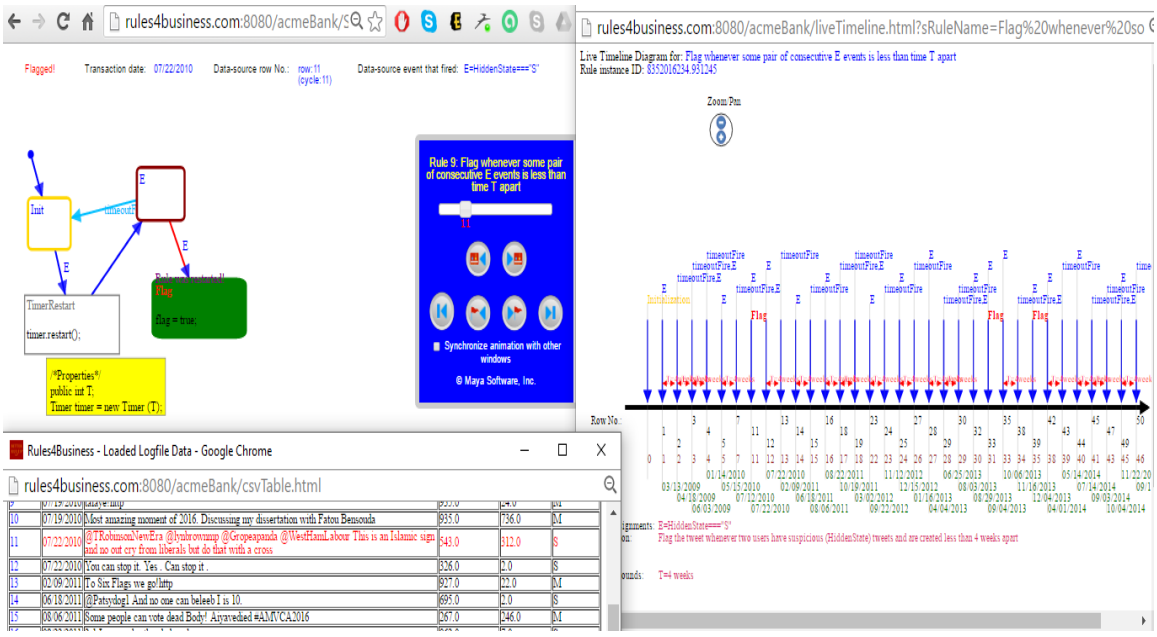


Figure 21. Success Flag for Rule 9 in R4B

E. STANDARD STATEROVER RULE CREATION AND CODE GENERATION

The StateRover provides detection of behavioral patterns by using deterministic UML based statechart patterns. StateRover extends the statechart based notation by combining statechart diagrams, Java action language, and the built-in Boolean flag *bSuccess.*

The StateRover is referred by this thesis because it is used as part of the code generation process; the code generator referred to in section H does so using code generated by the StateRover; therefore, we converted out R4B diagrams to StateRover diagrams. If you do not want to read the details about StateRover, please skip to the next section.

In this phase, R4B diagrams are converted to StateRover diagrams. For each R4B rule, we created a corresponding StateRover statechart-assertion (Figure 22). Statechart assertions start with an initial state. Events are the transitions between states. The final state is the Flag state that shows whether assertion fails or succeeds. If the StateRover

reaches Flag state, it assigns a *false* value to the *bSuccess* Boolean variable, meaning that the assertion discovered a flagged scenario.

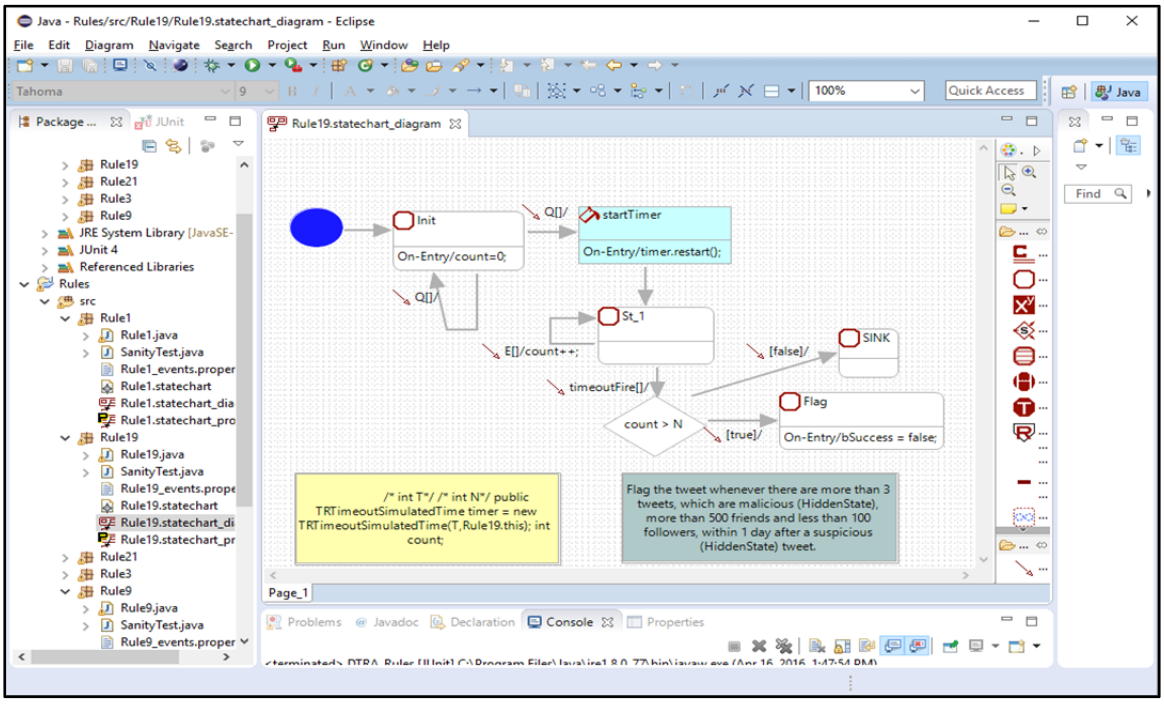


Figure 22. Rule-19 Statechart Diagram in StateRover

In the StateRover, automatic code generation requires that only two arguments be created. The first one is the rules_events.properties files for each rule. These files are simple text files; Figure 23 shows an example. They contain text that we already used in the R4B phase (see the Events and Limits&Bounds sections in Table 5).

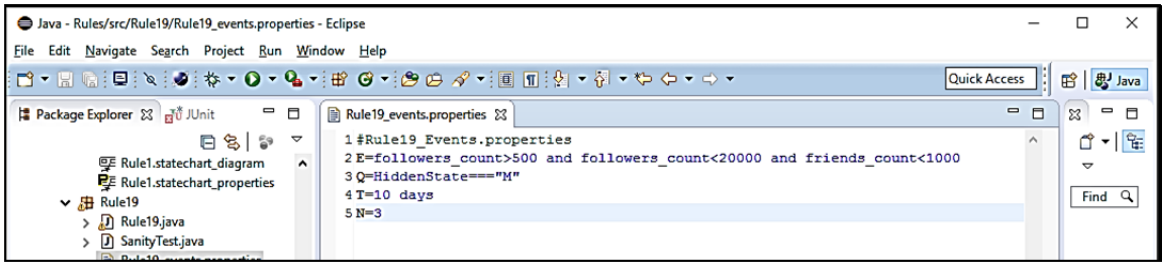


Figure 23. Rules19_events.properties File

The StateRover implements a two-step process to perform validation for checking the R4B diagrams are drawn accurately in the StateRover. In the first step it generates Java code by simply saving our statechart diagrams. In the second step, we need a JUnit test to execute to assure that the StateRover has the same behaviors for each statechart assertions as their R4B counterparts. All JUnit sanity test codes are in Appendix A. Figure 24 shows that all sanity tests run correctly.

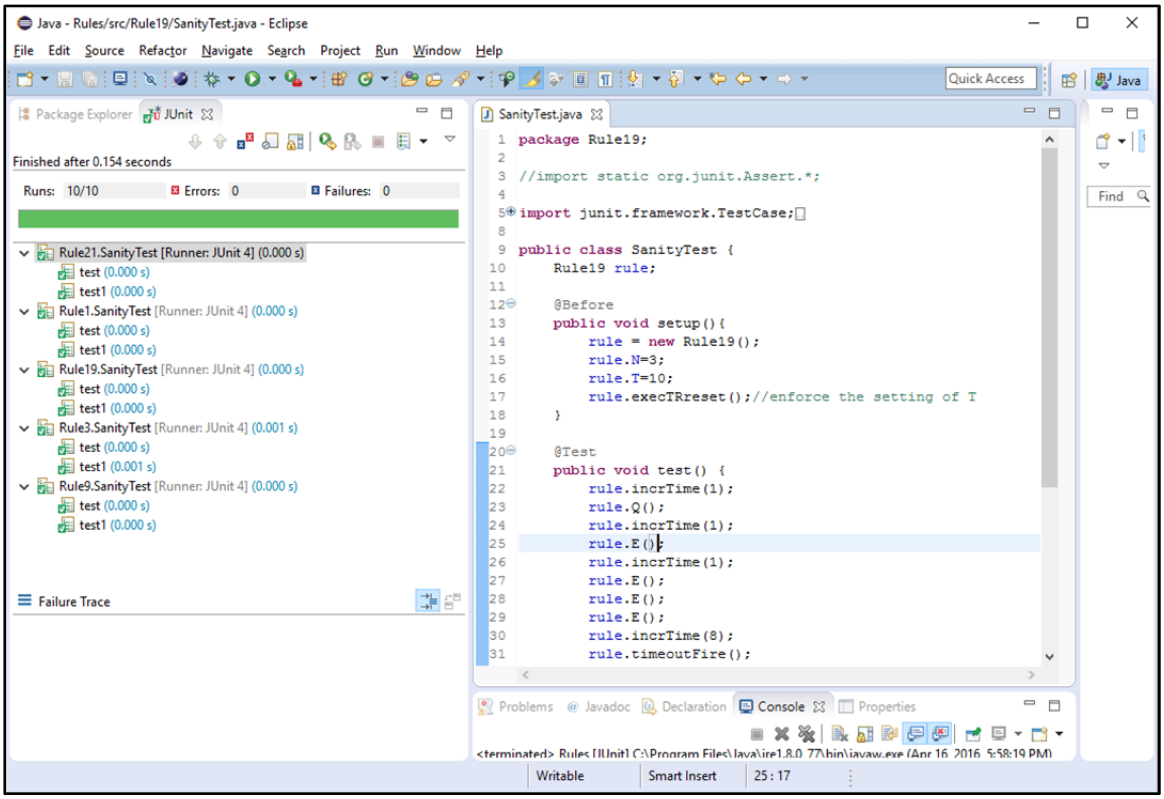


Figure 24. Sanity Tests Run

F. LEARNING PHASE FOR HMM

In the learning phase, we need to add a HiddenState column to our spreadsheet as column 7. Each tweet can be classified within three categories: malicious (M), suspicious (S), or benign (B). In this part, in order to populate the learning-phase spreadsheet, we act as a tweet classification expert. Table 5 defines the rule used for specifying the values of HiddenState. In order to determine the values of the HiddenState column, a human

operator uses followers_count, friends_count, user_verified, and geo_enabled columns. Figure 25 shows a snippet of our learning phase spreadsheet. HMM creation and the learning algorithm are explained in Section G.

Table 5. The Rule to Determine HiddenState in Learning Phase

Columns	Observation	Action
followers_count:	If the number of followers is between 500 and 20000	add 1 to total
friends_count:	If the number of friends is <1000	add 1 to total
user_verified	If the user is not verified	add 1 to total
geo_enabled	If the account does not enable geolocation	add 1 to total
If the total is 4 : assign M (malicious) 2-3 : assign S (suspicious) 0-1 : assign B (benign)		

followers_count	friends_count	user_verified	geo_enabled	HiddenState
2946	1506	FALSE	TRUE	**S**
219	740	FALSE	TRUE	**S**
204	149	FALSE	TRUE	**S**
265880	420	TRUE	TRUE	**B**
1226	828	FALSE	TRUE	**S**
695	2	FALSE	FALSE	**M**
204	149	FALSE	FALSE	**S**
580	1	FALSE	FALSE	**M**
935	24	FALSE	FALSE	**M**

Figure 25. The Population of the HiddenState Column

G. GENERATING THE HIDDEN MARKOV MODEL (HMM)

An HMM is a statistical model in which the state is not fully visible. However, the observable, which depends on state, is visible. The model determines one of the possible outputs by looking at the sequence of observables [19]. It provides a way to capture patterns that are essential to for making more accurate decisions.

In our thesis, the hidden states are the categorization of tweets. We already determined our hidden column according to the rules in the learning phase (Table 5). The spreadsheet, including populated hidden column and other visible data used in R4B rules instances, is our learning-phase spreadsheet (csv file). The learning phase csv also contains an indication of the initial state. The learning phase spreadsheet separates data into two types: visible data (friends_count, followers_count, text, and user_created_at) and hidden data (HiddenState). Let *v*, *h*, and *N* represent visible data, hidden data, and total number of rows, respectively. Also, let h_i and v_i be the values of the hidden and visible columns in row i. As Drusinsky states in [23], our HMM learns the probability as shown below:

- The probability of transition between states is obtained by dividing the number of specific transition to *N*-1 (that is the total number of transition in spreadsheet). For example, we have 40 transitions from suspicious (S) to malicious (M) in *h* and *N* is 81. So, the probability of HMM transition for *S->M* is #(S -> M)/(N-1) that equals *0.5*.

- Suppose that for a given row i, v_i is k and h_i is M. The probability of an observable k being emitted in a hidden state M is calculated by dividing the number of times when a row satisfies k and M with N.

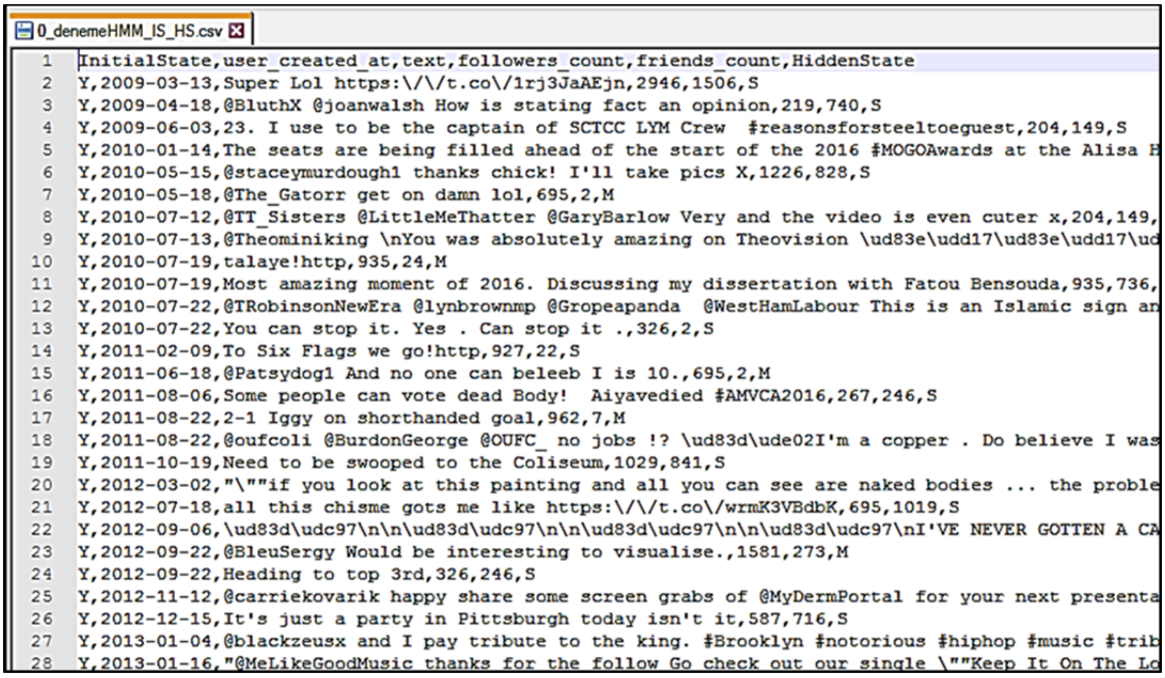
- Initial state probability distribution is the proportional number of times a hidden state is marked as an initial state. For instance, if we have two states of three marked as initial states then the initial state probability distribution is [0.5, 0.5, 0].

While R4B supports an event such as *followers_count<500*, being an infinite set of possible observables, a typical HMM operates on a relatively small number of observables. Therefore, we need to quantize our values with corresponding value range such as *followers_countLT500* (number of friends is less than 500). Table 6 indicates how columns user_created_at, friends_count, and followers_count will be quantized. The Python codes for quantization are presented in Appendix B. Quantization enables our toolset to map generic names like P, Q, and R.

The HMM generator needs a column named HiddenState. As we showed in section F, we played the role of an expert and filled the cells for HiddenState column. A snapshot of the final learning-phase csv file is shown in Figure 26.

Table 6. Quantization of Columns.

Columns	Values	Description
user_created_at	new old	If the account is created more than two years ago, this account is OLD. Otherwise it is NEW.
friends_count	friends_countLT500 friends_count500to1000 friends_countGT1000	
followers_count	followers_countLT500 followers_count500to20000 followers_countGT20000	



The first column for each row is Y that is just a special column indicating Initial State.

Figure 26. Learning-Phase CSV File.

The last work in this step is to run a command for generating an hmm.json file that includes the HMM in JSON (JavaScript Object Notation) (Figure 27).

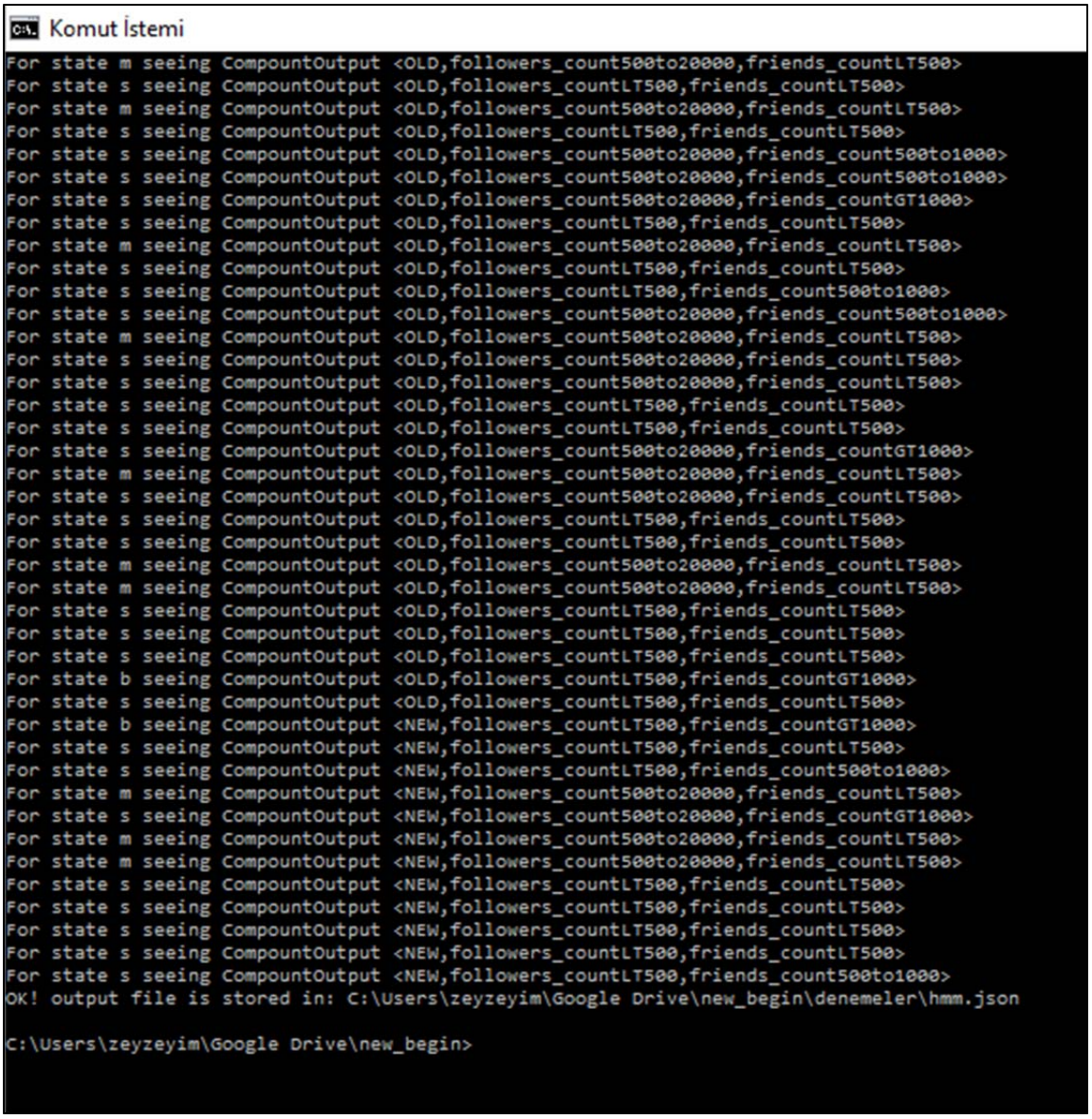


Figure 27. Command Prompt Run for Quantization

H. GENERATING SPECIAL JAVA CODE FOR PROBABILISTIC RUNTIME MONITORING

In this step, we create a new Java project including automated Java codes and sanity tests (Figure 28). Sanity tests validate the generated Java codes running correctly. Appendix C shows the sanity tests of Rule1_DTRA, Rule3_DTRA, Rule9_DTRA, Rule19_DTRA and Rule21_DTRA java files.

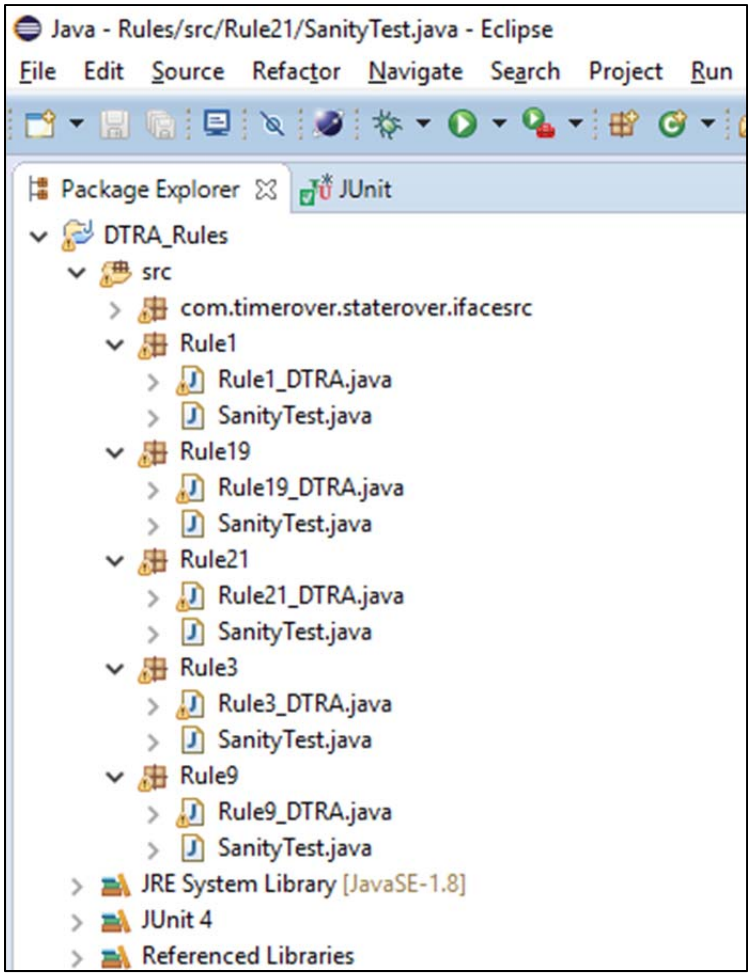


Figure 28. DTRA_Rules.

The special Java code for probabilistic RM implements an algorithm indicated in [23]. This algorithm uses an input sequence in the form of a two-tuple list, such as Input={K_1,P_1},{K_2,P_2}, {K_3,P_3}…{K_N,P_N}. K_i is either a visible event (i.e., friends_count, followers_count columns) or a hidden one (i.e., HiddenState column). P_i is the probability of distribution (POD) of K_i. The POD of a visible event is 1; the POD of a hidden event is taken from the results of running the alpha method on the HMM (the Alpha Method in Section I).

As pointed out in [23], the run time evaluation of an assertion consists of a collection of objects called configurations. We label a collection as Col and a configuration as Conf. Each Conf has a present state PS(Conf) and a probability value

called P(Conf) being the probability of the assertion being in state configuration Conf. In the start-up, there is a single configuration Conf whose with P(Conf)=1. Given and event K_i whose Pi is less than 1 (i.e., K_i is hidden), the Conf respond with the pairs, {Si, Pi}, with two configurations called Conf1 and Conf2. The probabilities and states of Conf1 and Conf2 are calculated as follows:

P(Conf1)=P(Conf)*Pi and P(Conf2)=1-P(Conf1)

PS(Conf1) is the following state decided by transition, if event fired. Otherwise, PS(Conf) assigned the PS(Conf2).

Note that two or more configurations that share the same present states are combined into one configuration as $Conf_{combined}$ by summing all participating P'(Conf) probabilities.

A statechart assertions declares the probability of a violation of its corresponding requirements, also known as probability of failure (POF) [23], being the sum of all P(Conf) for all Conf that reach the StateRover error (R4B flag) state.

I. RUNTIME MONITORING

1. The Alpha Method

A critical part of the novel RM process used in this thesis is the execution of the HMM alpha-method detailed in the sequel. The outcome of this step is a file call alpha.json (Figure 29).

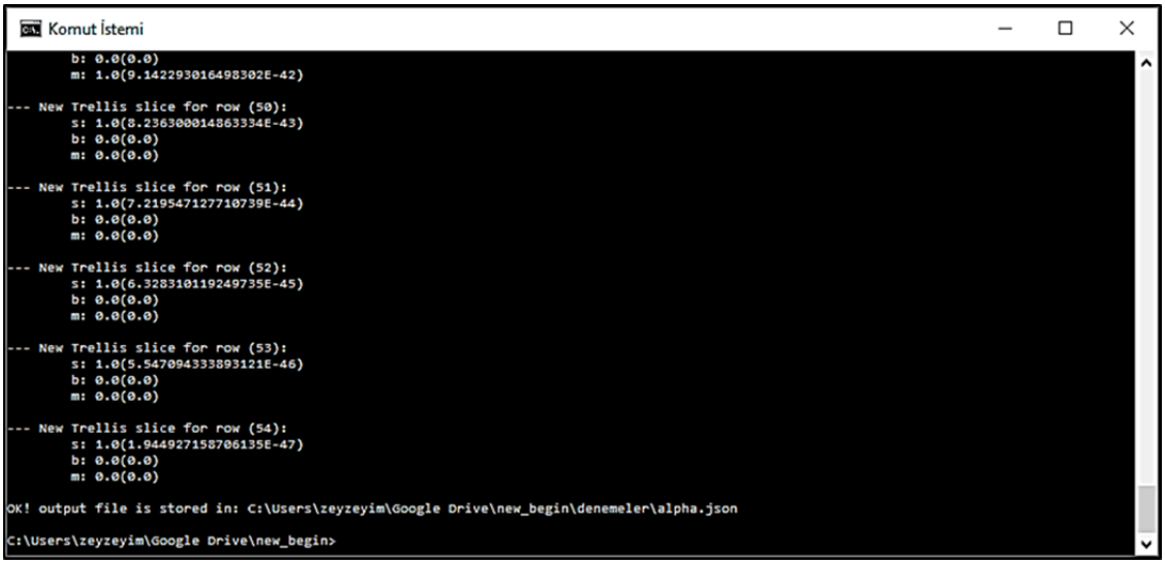


Figure 29. Generating alpha.json File.

According to [23], the alpha method calculates the HMM's POD over time, given the input csv file (Figure 30). More specifically, for every time slot t (i.e., for every row of the csv file), each HMM state *s* is assigned an alpha value $\alpha_s(t)$ being the probability of the HMM being in state *s* at time *t*.

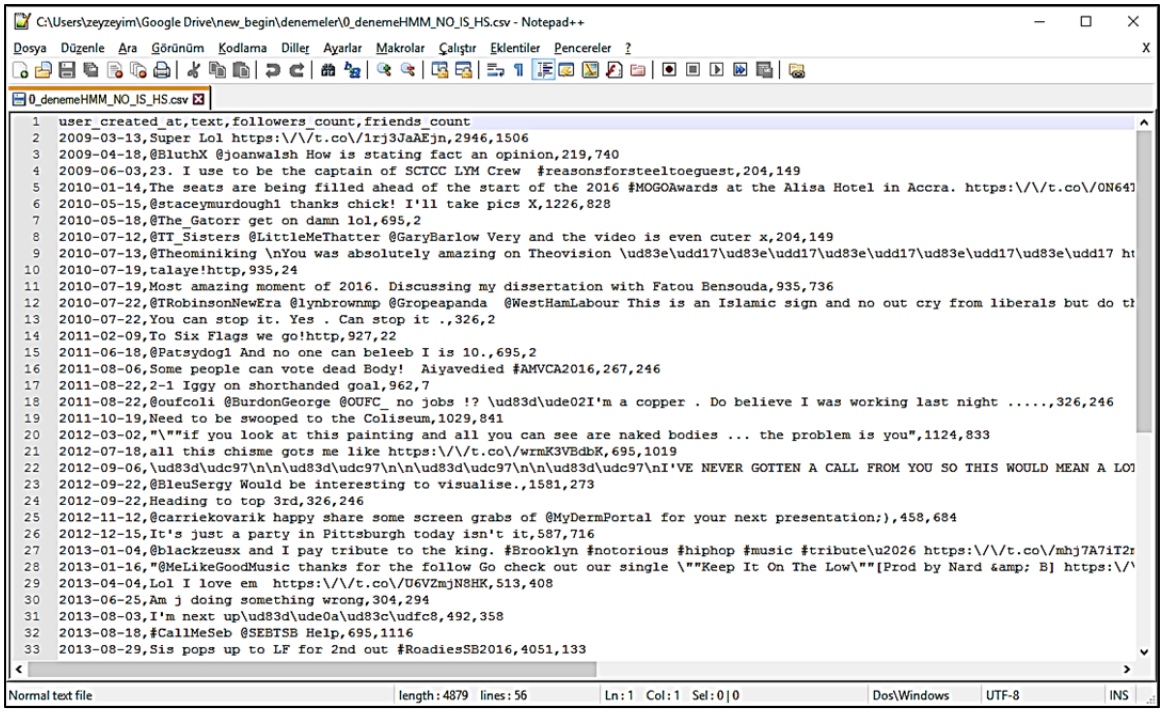
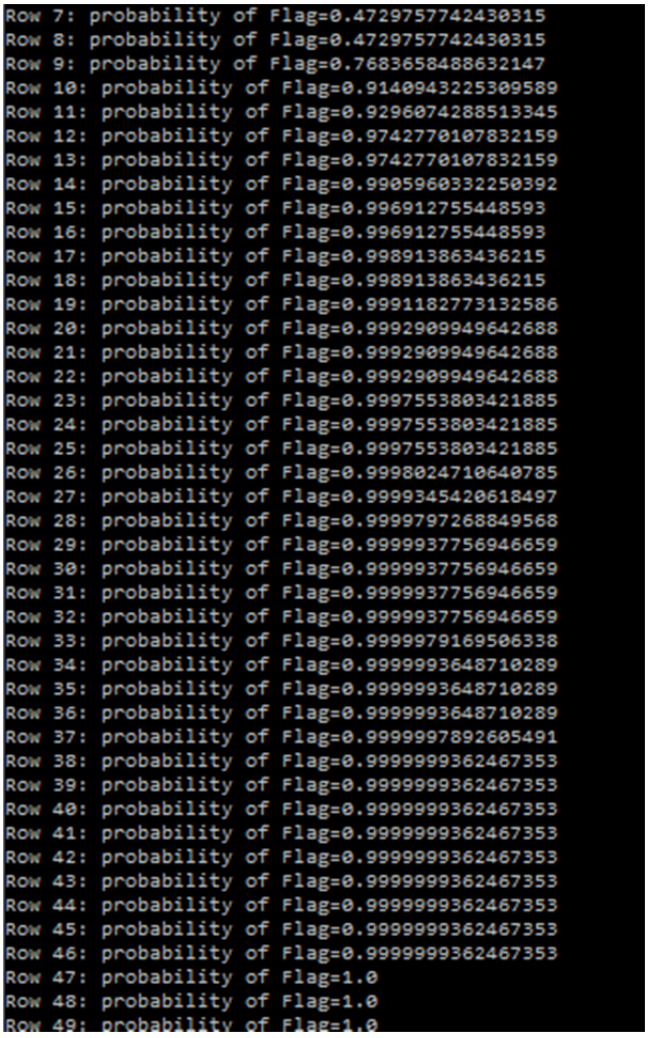


Figure 30. Runtime CSV File.

2. Probability of Flag States

Each row of each rule has a probability value in the range 0-1. This probability represents the likeliness of reaching the flag state. Figure 31 shows a list of probabilities for Rule 3. For example, while row 7 has a 47% probability to reach a flag state, this probability for row 47 is 100%. The tool presents an effective way to deal with malicious users and tweets. Because defining a threshold and analyzing the data up to this threshold can save time and effort.



A list of probability values; one per cycle (CSV file row) is the probability of the monitor reaching the Flag state in that cycle.

Figure 31. Runtime Monitoring Rule 3

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CONCLUSIONS

A. SUMMARY

In this thesis, we demonstrated a new technique to perform RM with hidden data. The purpose of the thesis is to determine whether using such technique for the detection of malicious tweets/users has the potential of detecting patterns of interest more efficiently than done so far.

This new technique uses a powerful tool, which is more effective than others are since it has the capability of handling datasets including non-observable data. In addition, this technique uses English specifications as the starting point, yet caters for unambiguity (using underlying formal specifications) and visual debugging; for example, an English starting point rule is “Flag whenever event Q occurs fewer than N times between events P and R.”

The technique can be used for pattern detection in many different domains, such as detection of fraudulent credit card transactions, traffic light controllers, automated border security and warning systems, detection of malicious email, tweet, and messages, etc.

Determining the NL assertions and converting them into corresponding formal specification language is a problematic area in software engineering. In our technique, UML-based statecharts offer a low learning curve and are very intuitive and simple. Automated code generation in the StateRover phase makes the tool a technique combining validation and monitoring of data. Simple implementation, domain independency, and automated code generation with runtime monitoring are the features that differentiate it from other tools.

In social media, although there is voluminous data flow, it is still possible to create an effective system that can detect malicious activities in a brief time and provide situational awareness. The important part of the work for a reliable system is to specify event sequences that indicate malicious activity. Considering event sequences allow the

system to deduce more precise inferences. In this thesis, we specify some event patterns indicating malicious activities according to [6].

Finally, there is no magic system to detect malicious content in Twitter or other social media platforms. However, there are some approaches to create new systems providing better situational awareness like this technique.

B. FUTURE WORK

Social media analysis is very popular and there are many opportunities for extending the scope of this thesis.

In this thesis, we use the relationship between malicious users and six attributes of tweets. These attributes, which are only a small part of all available attributes, are the creation date of the account, the number of friends and followers, enabling geolocation, verified account, and text part. It is possible to add more attributes for more accurate results. What is necessary is to find different indicators of malicious content and user behavior, then use them with related attributes in the work flow.

The malicious content and users can be subclassified, such as “terrorist” and “fraudulent behavior.” Because these can have different indicators, future studies could focus on either category.

APPENDIX. SCREENSHOTS FROM WORK FLOW

C. JUNIT SANITY TESTS

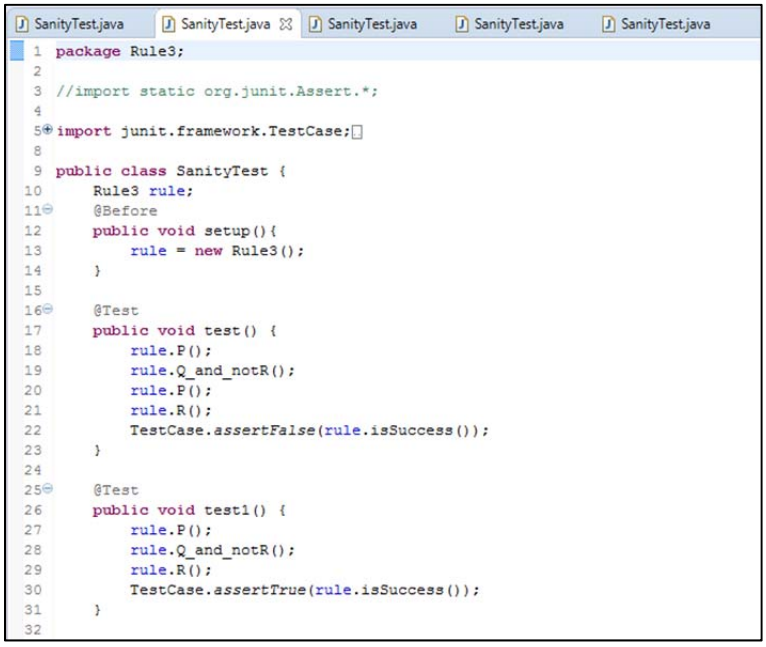
1. Rule-1

SanityTest.java | SanityTest.java | SanityTest.java | SanityTest.java | SanityTest.java

```
1  package Rule1;
2
3  //import static org.junit.Assert.*;
4
5  import junit.framework.TestCase;
8
9  public class SanityTest {
10     Rule1 rule;
11
12     @Before
13     public void setup(){
14     rule = new Rule1();
15     }
16
17     @Test
18     public void test() {
19         rule.P();
20         TestCase.assertFalse(rule.isSuccess());
21     }
22     @Test
23     public void test1() {
24         rule.timeoutFire();
25         TestCase.assertTrue(rule.isSuccess());
26     }
27
28 }
29
30
```

Figure 32. Rule 1 Sanity Test.

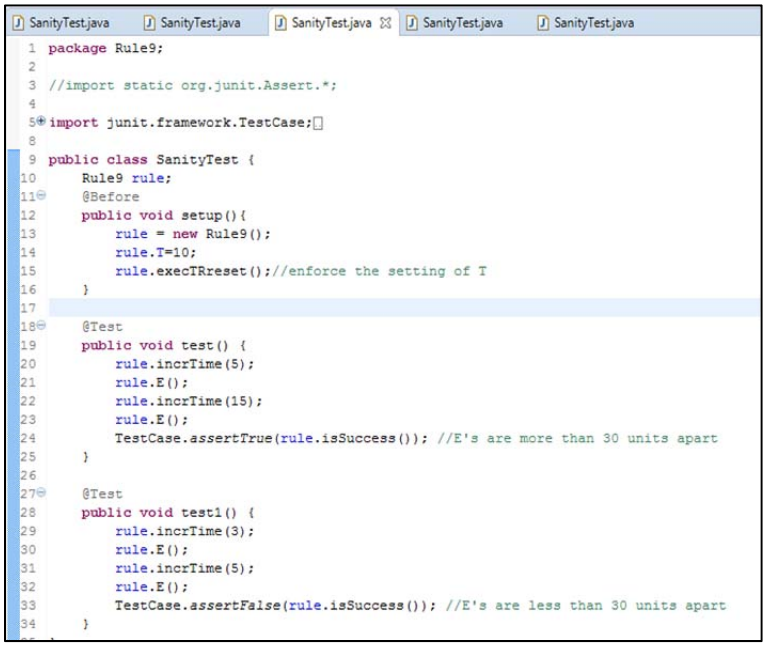
2. Rule-3




```java
package Rule3;

//import static org.junit.Assert.*;

import junit.framework.TestCase;

public class SanityTest {
    Rule3 rule;
    @Before
    public void setup(){
        rule = new Rule3();
    }

    @Test
    public void test() {
        rule.P();
        rule.Q_and_notR();
        rule.P();
        rule.R();
        TestCase.assertFalse(rule.isSuccess());
    }

    @Test
    public void test1() {
        rule.P();
        rule.Q_and_notR();
        rule.R();
        TestCase.assertTrue(rule.isSuccess());
    }

```

Figure 33. Rule 3 Sanity Test.

3. Rule-9




```java
package Rule9;

//import static org.junit.Assert.*;

import junit.framework.TestCase;

public class SanityTest {
    Rule9 rule;
    @Before
    public void setup(){
        rule = new Rule9();
        rule.T=10;
        rule.execTRreset();//enforce the setting of T
    }

    @Test
    public void test() {
        rule.incrTime(5);
        rule.E();
        rule.incrTime(15);
        rule.E();
        TestCase.assertTrue(rule.isSuccess()); //E's are more than 30 units apart
    }

    @Test
    public void test1() {
        rule.incrTime(3);
        rule.E();
        rule.incrTime(5);
        rule.E();
        TestCase.assertFalse(rule.isSuccess()); //E's are less than 30 units apart
    }
```

Figure 34. Rule 9 Sanity Test.

4. Rule-19

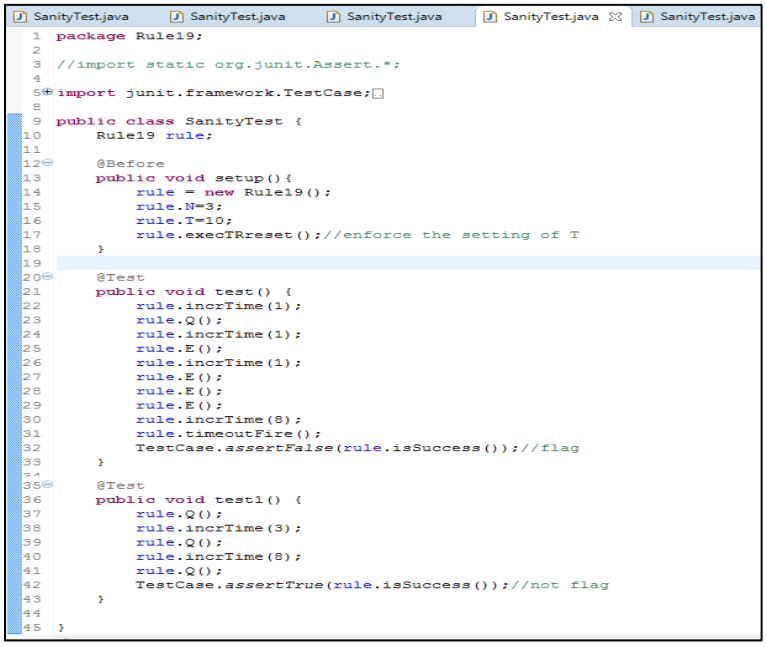


Figure 35. Rule 19 Sanity Test.

5. Rule-21

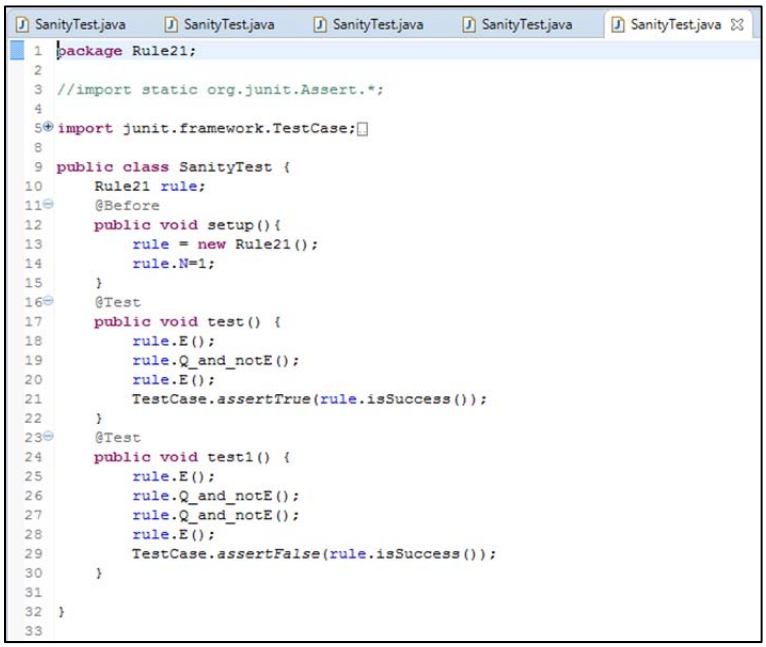


Figure 36. Rule 21 Sanity Test.

D. PYTHON QUANTIZATION SCRIPTS

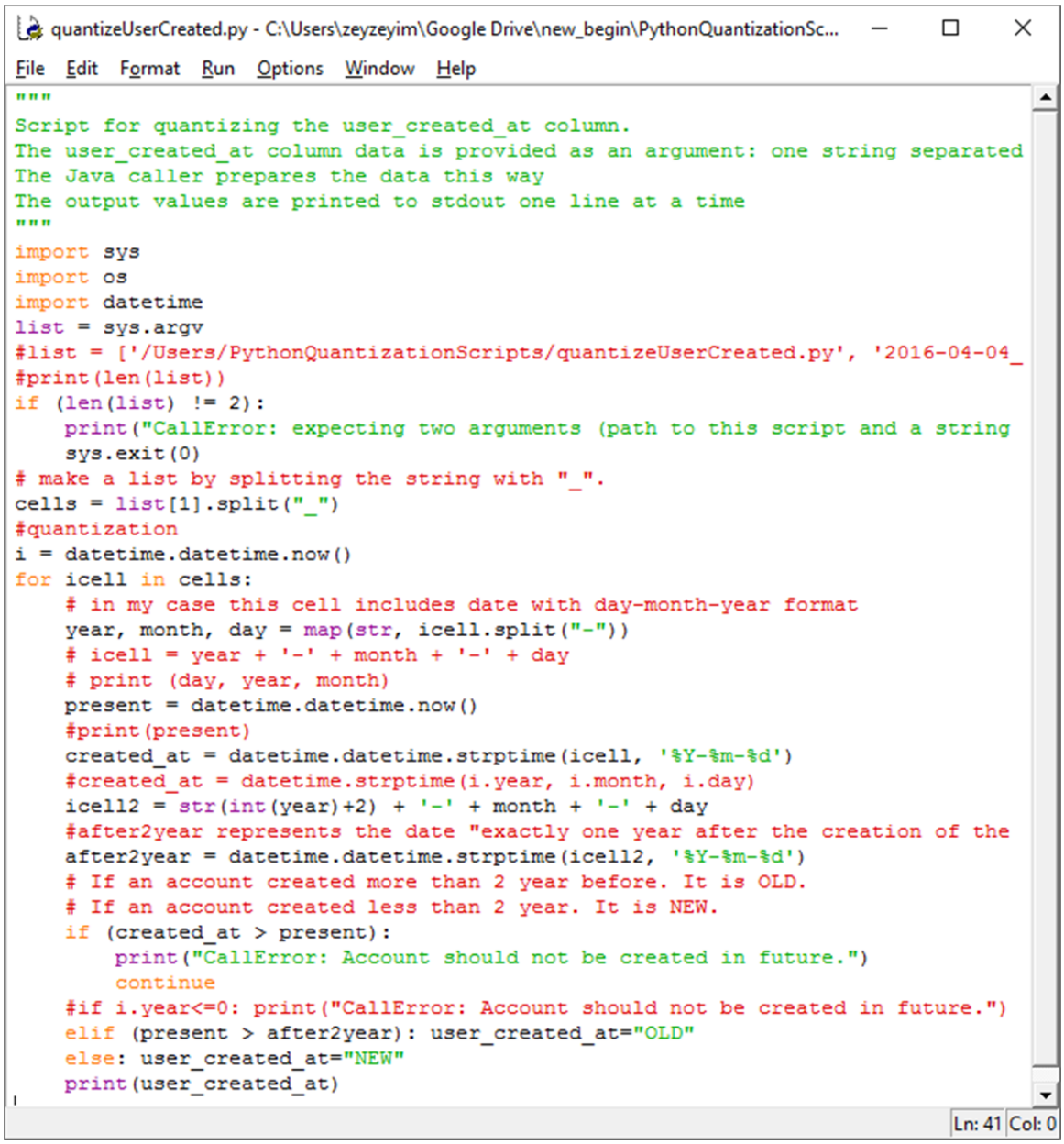


Figure 37. Quantize user_created_at Column.

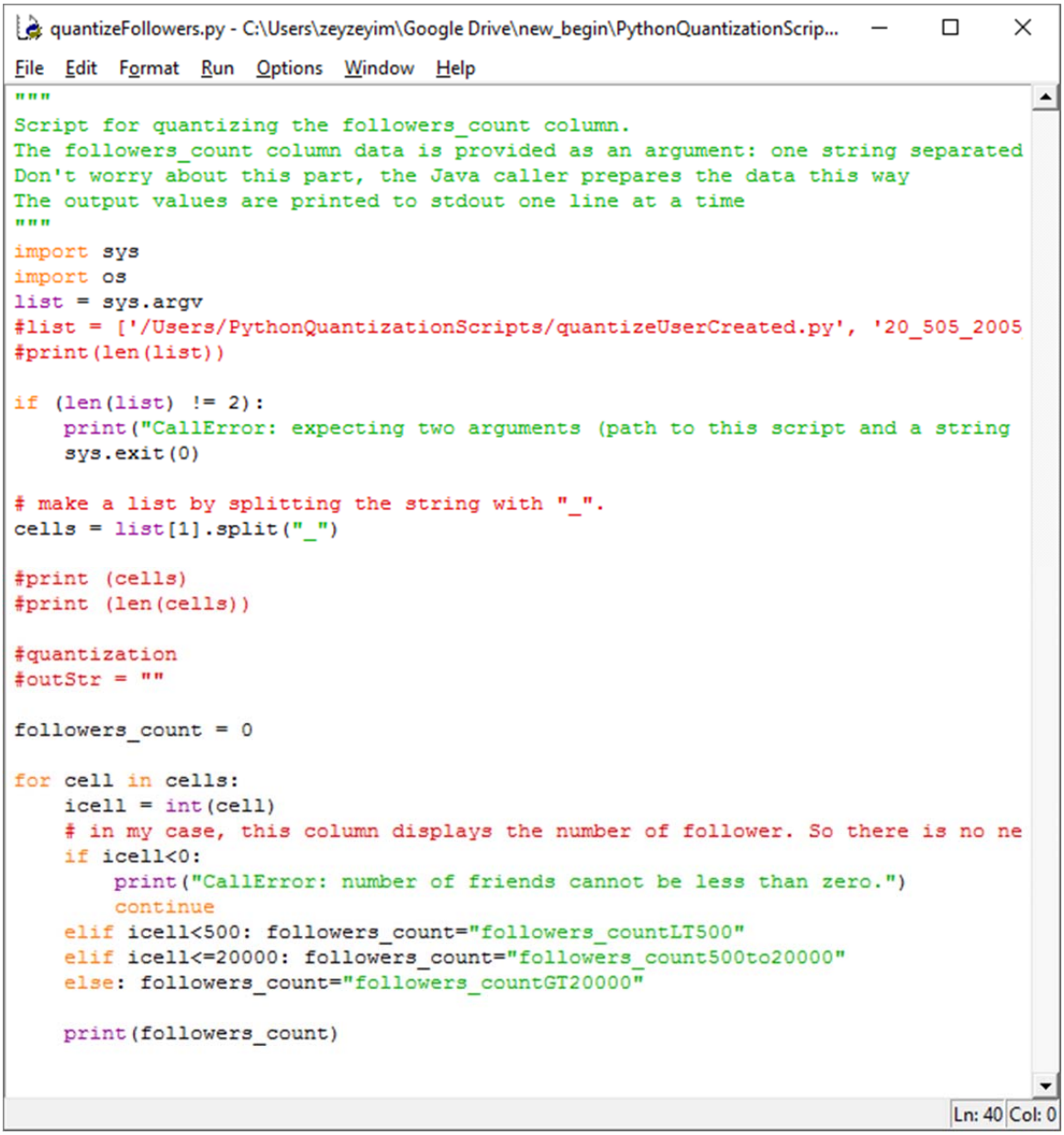


Figure 38. Quantize followers_count Column.

Figure 39. Quantize friends_count Column.

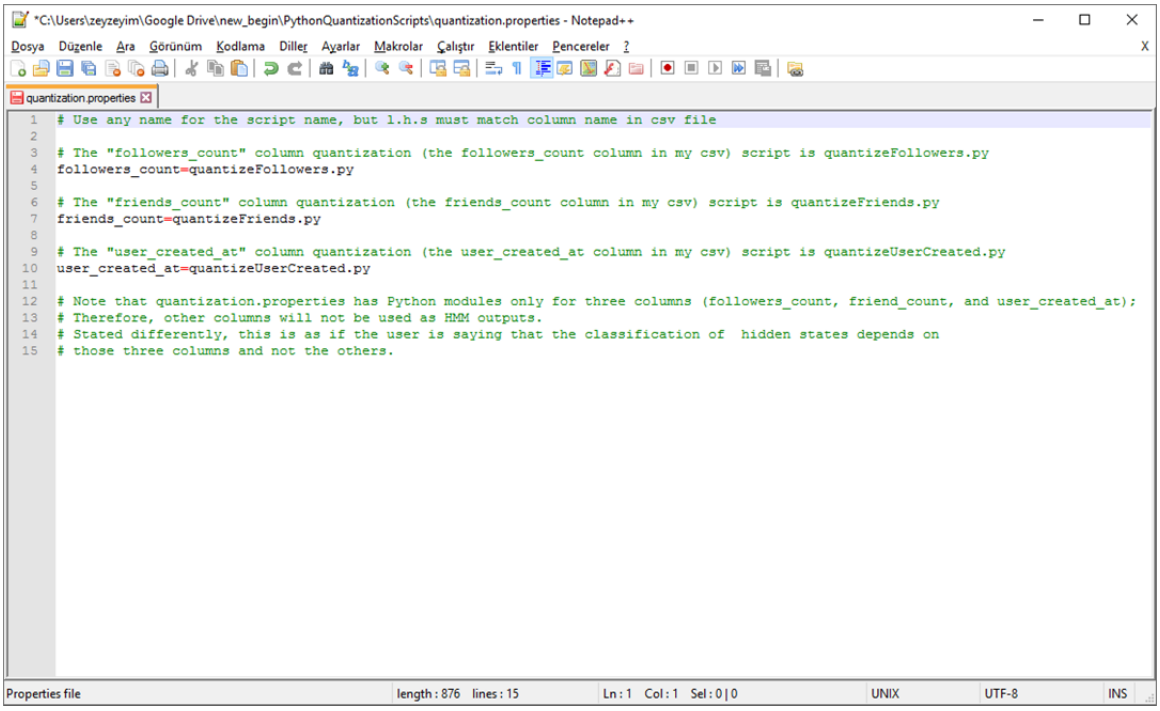


Figure 40. Quantization Properties File.

E. SANITY TESTS FOR PROBABILISTIC RUNTIME VERIFICATION

1. Rule-1 Sanity Test for DTRA_Rules

SanityTest.java

```
1  package Rule1;
2  //import static org.junit.Assert.*;
3  import junit.framework.TestCase;
4  import org.junit.Before;
5  import org.junit.Test;
6  public class SanityTest {
7      Rule1_DTRA rule;
8      @Before
9      public void setup(){
10     rule = new Rule1_DTRA();
11     }
12     @Test
13     public void test() {
14         rule.P(1.0);
15         double d = rule.getProbabilityOfSuccess();
16         System.out.println("d="+d);
17         TestCase.assertFalse(rule.isSuccess());
18         // probability 0 of success means probability 1 of flagging -- FLAG
19         TestCase.assertEquals(1-d, 1.0);
20     }
21     @Test
22     public void test1() {
23         rule.timeoutFire(1.0);
24         double d = rule.getProbabilityOfSuccess();
25         System.out.println("in test: d="+d);
26         TestCase.assertTrue(rule.isSuccess());
27         // probability 1 of success means probability 0 of flagging  -- NO FLAG
28         TestCase.assertEquals(1-d, 0.0);
29     }
30
```

Figure 41. Sanity Test for Rule1_DTRA.

2. Rule-3 Sanity Test for DTRA_Rules

SanityTest.java

```
1  package Rule3;
2  //import static org.junit.Assert.*;
3  import junit.framework.TestCase;
4  import org.junit.Before;
5  import org.junit.Test;
6  public class SanityTest {
7      Rule3_DTRA rule;
8      @Before
9      public void setup(){
10         rule = new Rule3_DTRA();
11     }
12     @Test
13     public void test() {
14         rule.P(1.0);
15         rule.R(1.0);
16         double d = rule.getProbabilityOfSuccess();
17         System.out.println("in test: d="+d);
18         TestCase.assertFalse(rule.isSuccess());
19         // probability 0 of success means probability 1 of flagging
20         TestCase.assertEquals(1-d, 1.0);
21     }
22     @Test
23     public void test1() {
24         rule.P(1.0);
25         rule.Q_and_notR(1.0);
26         rule.P(1.0);
27         rule.Q_and_notR(1.0);
28         rule.R(1.0);
29         double d = rule.getProbabilityOfSuccess();
30         System.out.println("in test: d="+d);
```

Figure 42. Sanity Test for Rule3_DTRA.

3. Rule-9 Sanity Test for DTRA_Rules

*SanityTest.java

```
 8      public void setup(){
 9          rule = new Rule9_DTRA();
10          rule.T=10;
11          rule.execTRreset();//enforce the setting of T
12      }
13      @Test
14      public void test() {
15          rule.incrTime(5);
16          rule.E(1.0); //add probability 1 to events
17          rule.incrTime(15);
18          rule.E(1.0); //add probability 1 to events
19          double d = rule.getProbabilityOfSuccess();
20          System.out.println("in test: d="+d);
21          //E's are more than 30 units apart
22          TestCase.assertTrue(rule.isSuccess());
23          // probability 1 of success means probability 0 of flagging
24          TestCase.assertEquals(1-d, 0.0);
25      }
26      @Test
27      public void test1() {
28          rule.incrTime(3);
29          rule.E(1.0); //add probability 1 to events
30          rule.incrTime(5);
31          rule.E(1.0); //add probability 1 to events
32          double d = rule.getProbabilityOfSuccess();
33          System.out.println("in test: d="+d);
34          //E's are less than 30 units apart
35          TestCase.assertFalse(rule.isSuccess());
36          // probability 0 of success means probability 1 of flagging
37          TestCase.assertEquals(1-d, 1.0);
```

Figure 43. Sanity Test for Rule9_DTRA.

4. Rule-19 Sanity Test for DTRA_Rules

*SanityTest.java

```
7       Rule19_DTRA rule;
8       @Before
9       public void setup(){
10          rule = new Rule19_DTRA();
11          rule.N=2;
12          rule.T=10;
13          rule.execTRreset();//enforce the setting of T
14      }
15      @Test
16      public void test() {
17          rule.incrTime(1);
18          rule.Q(1.0);    //add probability 1 to events
19          rule.incrTime(1);
20          rule.E(1.0);
21          rule.incrTime(1);
22          rule.E(1.0);
23          rule.E(1.0);
24          rule.E(1.0);
25          rule.incrTime(8);
26          rule.timeoutFire(1.0);
27          double d = rule.getProbabilityOfSuccess();
28          System.out.println("d="+d);
29          TestCase.assertFalse(rule.isSuccess());
30          // probability 0 of success means probability 1 of flagging -- FLAG
31          TestCase.assertEquals(1-d, 1.0);
32      }
33      @Test
34      public void test1() {
35          rule.Q(1.0);
36          rule.incrTime(3);
```

Figure 44. Sanity Test for Rule19_DTRA.

5. Rule-21 Sanity Test for DTRA_Rules

SanityTest.java

```
6  public class SanityTest {
7      Rule21_DTRA rule;
8      @Before
9      public void setup(){
10         rule = new Rule21_DTRA();
11         rule.N=1;
12     }
13     @Test
14     public void test() {
15         rule.E(1.0);
16         rule.Q_and_notE(1.0);
17         rule.E(1.0);
18         double d = rule.getProbabilityOfSuccess();
19         System.out.println("in test: d="+d);
20         TestCase.assertTrue(rule.isSuccess());
21         // probability 1 of success means probability 0 of flagging
22         TestCase.assertEquals(1-d, 0.0);
23     }
24     @Test
25     public void test1() {
26         rule.E(1.0);
27         rule.Q_and_notE(1.0);
28         rule.Q_and_notE(1.0);
29         rule.E(1.0);
30         double d = rule.getProbabilityOfSuccess();
31         System.out.println("in test: d="+d);
32         TestCase.assertFalse(rule.isSuccess());
33         // probability 0 of success means probability 1 of flagging
34         TestCase.assertEquals(1-d, 1.0);
35     }
```

Figure 45. Sanity Test for Rule21_DTRA.

F. COMMANDS IN CMD

Table 7. Commands.

Action	Command
Generating hmm.json file	java -jar dtrahmm.jar denemeler\0_denemeHMM_IS_HS.csv PythonQuantizationScripts
	Argument-1: learning phase csv file. Argument-2: folder of quantization.properties files
Run the alpha method	java -jar dtraalpha.jar denemeler\0_denemeHMM_NO_IS_HS.csv denemeler\hmm.json PythonQuantizationScripts
	Argument-1: runtime csv file. Argument-2: hmm.json file Argument-3: folder of quantization.properties files
Runtime monitoring	java -jar dtrarm.jar denemeler\0_denemeHMM_NO_IS_HS.csv denemeler\alpha.json DTRA_Rules.jar Rule3_DTRA Rules\bin\Rule3\Rule3_events.properties
	Argument-1: runtime csv file Argument-2: path to alpha.json file Argument-3: path to DTRA_Rules.jar file Argument-4: the rule we want to monitor Argument-5: path to events.properties files
Color code for arguments	Argument-1Argument-2Argument-3Argument-4

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] S. Kemp. (2015, Jan. 21). Digital, social & mobile worldwide in 2015 [Online]. Available: http://wearesocial.com/uk/special-reports/digital-social-mobile-worldwide-2015

[2] Twitter. (2015, Nov. 27). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Twitter. Accessed: Nov. 29, 2015.

[3] List of social networking websites. (2015, Nov. 22). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/List_of_social_networking_websites. Accessed: Nov. 28, 2015.

[4] W. Richey. (2015, June 3). Terror on Twitter: How Islamic state uses social media to draw recruits (+video). *The Christian Science Monitor* [Online]. Available: http://www.csmonitor.com/USA/Justice/2015/0603/Terror-on-Twitter-How-Islamic-State-uses-social-media-to-draw-recruits-video

[5] D. Goldman. (2016, Feb. 5). Twitter goes to war against ISIS. CNN [Online]. Available: http://money.cnn.com/2016/02/05/technology/twitter-terrorists-isis/index.html.

[6] J. M. Berger and J. Morgan. (2015, Mar. 20). The ISIS Twitter Census: Defining and describing the population of ISIS supporters on Twitter [Online]. Available: http://www.brookings.edu/~/media/research/files/papers/2015/03/isis-twitter-census-berger-morgan/isis_twitter_census_berger_morgan.pdf.

[7] D. Drusinsky, J. B. Michael, and M.-T. Shing, "A visual tradeoff space for formal verification and validation techniques," *IEEE Systems Journal*, vol. 2, no. 4, pp. 513–519, Dec. 2008.

[8] T. Bures, P. Hnetynka, P. Kroha, and V. Simko (2012 Dec.) Requirement specifications using natural languages [Online]. Available: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2012-05.pdf.

[9] Formal specification. (2015, Sep. 9). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Formal_specification. Accessed: January 31, 2016.

[10] Formal Specification. [Online]. Available: http://c2.com/cgi/wiki?formalspecification. Accessed: December 26, 2016.

[11] Rules4business. (n.d.). D. Drusinsky. [Online]. http://www.rules4business.com/acmeBank/index.html. Accessed: December 26, 2015.

[12] Formal methods. (2016, Jan. 12). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Formal_methods. Accessed: February 20, 2016.

[13] D. Drusinsky, M.-T. Shing, and K. A. Demir, "Creating and validating embedded assertion Statecharts," *IEEE Distributed Systems Online*, vol. 8, no. 5, pp. 3–3, May 2007.

[14] J. J. Galinski, “Formal specifications for an electrical power grid system stability and reliability," M.S. thesis, Comp. Sci., Naval Postgraduate School, Monterey, CA, USA, 2015.

[15] Model checking. (2016, Jan. 12). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Model_checking. Accessed: February 01, 2016.

[16] Runtime verification. (2016, Feb. 1). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Runtime_verification. Accessed: February 08, 2016.

[17] Markov model. (2015, Dec. 8). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Markov_model. Accessed: February 08, 2016.

[18] Hidden Markov model. (2016, Feb. 3). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Hidden_Markov_model. Accessed: February 08, 2016.

[19] D. Drusinsky, "Behavioral and temporal pattern detection within financial data with hidden information," Naval Postgraduate School, Monterey, CA, Feb. 2012. [Online]. Available: http://calhoun.nps.edu/bitstream/handle/10945/24398/NPS-CS-12-002.pdf?sequence=3.

[20] D. Drusinsky, "Runtime monitoring and verification of systems with hidden information," *Innovations in Systems and Software Engineering*, vol. 10, no. 2, pp. 123–136, Sep. 2013.

[21] A. Moujahid. (2014, July 21) An introduction to text mining using Twitter streaming API and python [Online]. Available: http://adilmoujahid.com/posts/2014/07/twitter-analytics/

[22] M. Singh, B. Divya and S. Sanjeev. "Detecting malicious users in Twitter using classifiers," *Proceedings of the 7th International Conference on Security of Information and Networks - SIN '14*, 2014.

[23] D. Drusinsky, "Runtime monitoring and verification of systems with hidden information," *Innovations in Systems and Software Engineering*, vol. 10, no. 2, pp. 123–136, Sep. 2013.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California